

SCO UNIX[®] System V

Development System

Programmer's Reference

The Santa Cruz Operation, Inc.

Portions © 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989 Microsoft Corporation.

All rights reserved.

Portions © 1989 AT&T.

All rights reserved.

Portions © 1983, 1984, 1985, 1986, 1987, 1988, 1989 The Santa Cruz Operation, Inc.

All rights reserved.

No part of this publication may be reproduced, transmitted, stored in a retrieval system, nor translated into any human or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of the copyright owner, The Santa Cruz Operation, Inc., 400 Encinal, Santa Cruz, California, 95062, U.S.A. Copyright infringement is a serious matter under the United States and foreign Copyright Laws.

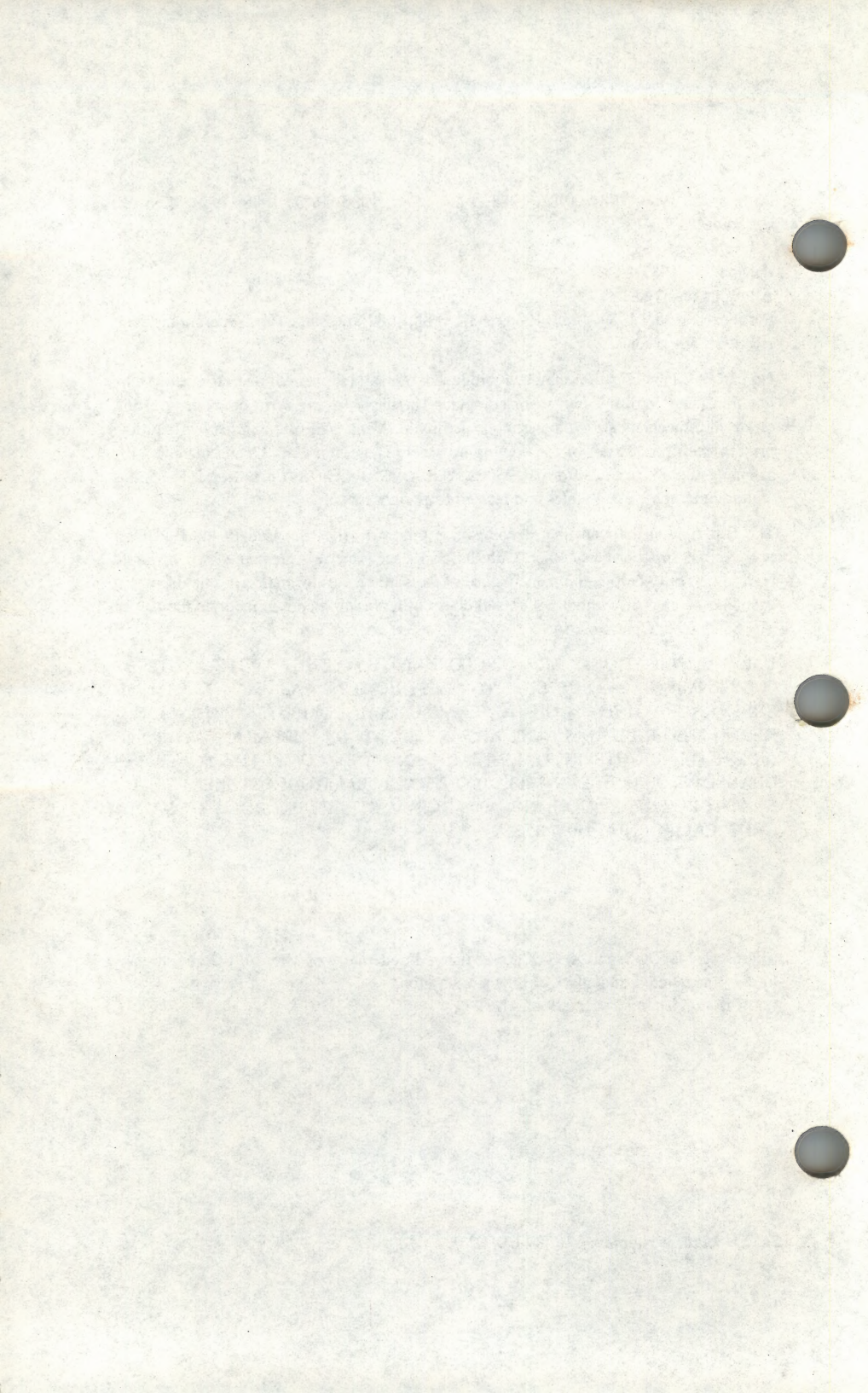
The copyrighted software that accompanies this manual is licensed to the End User only for use in strict accordance with the End User License Agreement, which should be read carefully before commencing use of the software. Information in this document is subject to change without notice and does not represent a commitment on the part of The Santa Cruz Operation, Inc.

USE, DUPLICATION, OR DISCLOSURE BY THE UNITED STATES GOVERNMENT IS SUBJECT TO RESTRICTIONS AS SET FORTH IN SUBPARAGRAPH (c) (1) OF THE COMMERCIAL COMPUTER SOFTWARE -- RESTRICTED RIGHTS CLAUSE AT FAR 52.227-19 OR SUBPARAGRAPH (c) (1) (ii) OF THE RIGHTS IN TECHNICAL DATA AND COMPUTER SOFTWARE CLAUSE AT DFARS 52.227-7013. "CONTRACTOR/ MANUFACTURER" IS THE SANTA CRUZ OPERATION, INC., 400 ENCINAL STREET, P.O. BOX 1900, SANTA CRUZ, CALIFORNIA, 95061, U.S.A.

Microsoft, MS-DOS, and XENIX are registered trademarks of Microsoft Corporation.

Intel is a registered trademark of Intel Corporation.

UNIX is a registered trademark of AT&T.



Preface

The complete UNIX Reference is actually divided into twelve parts and distributed as individual reference sections in the various volumes of the Operating and Development Systems. The following table lists the name, content, and location of each reference section.

Section	Description	Volume
ADM	Administrative Commands - used for system administration.	System Administrator's Reference
C	Commands - used with the Operating System.	User's Reference
CP	Programming Commands - used with the Development System.	Programmer's Reference
DOS	Routines - used with the Development System.	Developer's Guide
F	File Formats - description of various system files not defined in section M.	User's Reference
HW	Hardware device manual pages - information about hardware devices and device nodes.	System Administrator's Reference
K	Kernel routines - used for writing device drivers.	Device Driver Writer's Guide
M	Miscellaneous - information used for access to devices, system maintenance, and communications.	User's Reference
NSL	Network Services Library - used with the STREAMS System.	Developer's Guide

S	System Calls and Library Routines - available for C and assembly language programming.	Programmer's Reference
STR	STREAMS manual pages.	Developer's Guide
XNX	XENIX cross development manual pages.	Library Guide

In the manual pages, a given command, routine, or file is referred to by name and section. For example, the programming command “cc”, which is described in the Programming Commands (CP) section, is listed as *cc*(CP).

The alphabetized table of contents given on the following pages is a complete listing of all UNIX commands, system calls, library routines, and file formats. The permuted index, found at the end of the *User's Reference* and at the end of the *Programmer's Reference*, is useful in matching a desired task with the manual page that describes it.

Alphabetized List

Commands, Systems Calls, Library Routines and File Formats

300	300(C)	archive	archive(F)
300s	300(C)	as	as(CP)
450	450(C)	ascftime	ctime(S)
4014	4014(C)	ascii	ascii(M)
80387	80387(HW)	asctime	ctime(S)
FP_OFF	FP_OFF(DOS)	asin	trig(S)
FP_SEG	FP_OFF(DOS)	asktime	asktime(ADM)
Intro	intro(C)	assert	assert(S)
a64l	a64l(S)	assign	assign(C)
abort	abort(S)	at	at(C)
abs	abs(S)	atan	trig(S)
accept	accept(ADM)	atan2	trig(S)
acceptable_password		atcronsh	atcronsh(ADM)
.....	accept_pw(S)	atexit	atexit(DOS)
access	access(DOS)	atexit	atexit(S)
access	access(S)	atof	atof(S)
acct	acct(F)	atof	strtod(S)
acct	acct(S)	atoi	atof(S)
acctcms	acctcms(ADM)	atoi	strtol(S)
acctcom	acctcom(ADM)	atol	atof(S)
acctcon1	acctcon(ADM)	atol	strtol(S)
acctcon2	acctcon(ADM)	audit	audit(HW)
acctdisk	acct(ADM)	audit_close	audit(S)
acctdusg	acct(ADM)	auditcmd	auditcmd(ADM)
acctmerg	acctmerg(ADM)	auditd	auditd(ADM)
accton	acct(ADM)	audit_open	audit(S)
accton	accton(ADM)	audit_read	audit(S)
acctprc1	acctprc(ADM)	auditsh	auditsh(ADM)
acctprc2	acctprc(ADM)	authaudit	authaudit(S)
acctwtmpt	acct(ADM)	authcap	authcap(F)
acos	trig(S)	authcap	authcap(S)
adb	adb(CP)	authck	authck(ADM)
adb	adb(CP)	auths	auths(C)
adfnt	adfnt(ADM)	authsh	authsh(ADM)
admin	admin(CP)	autoboot	autoboot(ADM)
alarm	alarm(S)	awk	awk(C)
alloca	alloca(DOS)	backup	backup(ADM)
a.out	a.out(F)	backupsh	backupsh(ADM)
ar	ar(CP)	badtrk	badtrk(ADM)
ar	ar(F)	banner	banner(C)
ar	ar(XNX)	basename	basename(C)

batch	<i>at</i> (C)	chroot	<i>chroot</i> (ADM)
bc	<i>bc</i> (C)	chroot	<i>chroot</i> (S)
bcheckrc	<i>brc</i> (ADM)	chrtbl	<i>chrtbl</i> (M)
bdiff	<i>bdiff</i> (C)	chsize	<i>chsize</i> (S)
bdos	<i>bdos</i> (DOS)	ckpacct	<i>acctsh</i> (ADM)
bfs	<i>bfs</i> (C)	cleanque	<i>cleanque</i> (ADM)
boot	<i>boot</i> (HW)	cleantmp	<i>cleantmp</i> (ADM)
brc	<i>brc</i> (ADM)	clear	<i>clear</i> (C)
brk	<i>brk</i> (S)	_clear87	<i>_clear87</i> (DOS)
brkctl	<i>brkctl</i> (S)	clearerr	<i>ferror</i> (S)
bsearch	<i>bsearch</i> (S)	clock	<i>clock</i> (F)
cabs	<i>cabs</i> (DOS)	clock	<i>clock</i> (S)
cal	<i>cal</i> (C)	clone	<i>clone</i> (M)
calendar	<i>calendar</i> (C)	close	<i>close</i> (S)
calloc	<i>malloc</i> (S)	closedir	<i>directory</i> (S)
cancel	<i>lp</i> (C)	clri	<i>clri</i> (ADM)
captainfo	<i>captainfo</i> (ADM)	cmchk	<i>cmchk</i> (C)
cat	<i>cat</i> (C)	cmos	<i>cmos</i> (HW)
cb	<i>cb</i> (CP)	cmp	<i>cmp</i> (C)
cc	<i>cc</i> (CP)	codeview	<i>codeview</i> (CP)
cd	<i>cd</i> (C)	col	<i>col</i> (C)
cdc	<i>cdc</i> (CP)	coltbl	<i>coltbl</i> (M)
ceil	<i>floor</i> (S)	comb	<i>comb</i> (CP)
cfgetispeed	<i>cfspeed</i> (S)	comm	<i>comm</i> (C)
cfgetospeed	<i>cfspeed</i> (S)	compress	<i>compress</i> (C)
cflow	<i>cflow</i> (CP)	configure	<i>configure</i> (ADM)
cfsetispeed	<i>cfspeed</i> (S)	console	<i>console</i> (M)
cfsetospeed	<i>cfspeed</i> (S)	consoleprint	
cftime	<i>ctime</i> (S)	<i>consoleprint</i> (ADM)
cgets	<i>cgets</i> (DOS)	_control87 ..	<i>_control87</i> (DOS)
chargefee	<i>acctsh</i> (ADM)	conv	<i>conv</i> (CP)
chdir	<i>chdir</i> (DOS)	conv	<i>conv</i> (S)
chdir	<i>chdir</i> (S)	convert	<i>convert</i> (CP)
check_basic_data_structures		convkey	<i>mapkey</i> (M)
.....	<i>check_data</i> (S)	copy	<i>copy</i> (C)
checklist	<i>checklist</i> (F)	copydvagent	<i>getdvagent</i> (S)
checkmail	<i>checkmail</i> (C)	core	<i>core</i> (F)
checkque	<i>checkque</i> (ADM)	cos	<i>trig</i> (S)
chg_audit	<i>chg_audit</i> (ADM)	cosh	<i>sinh</i> (S)
chgrp	<i>chgrp</i> (C)	cp	<i>cp</i> (C)
chkshlib	<i>chkshlib</i> (CP)	cpio	<i>cpio</i> (C)
chmod	<i>chmod</i> (C)	cpio	<i>cpio</i> (F)
chmod	<i>chmod</i> (DOS)	cpg	<i>cpg</i> (CP)
chmod	<i>chmod</i> (S)	cprintf	<i>cprintf</i> (DOS)
chown	<i>chown</i> (C)	cprs	<i>cprs</i> (CP)
chown	<i>chown</i> (S)	cputs	<i>cputs</i> (DOS)

crash	<i>crash</i> (ADM)	dfscsk	<i>fsck</i> (ADM)
creat	<i>creat</i> (DOS)	dial	<i>dial</i> (ADM)
creat	<i>creat</i> (S)	dial	<i>dial</i> (S)
creatsem	<i>creatsem</i> (S)	dialcodes	<i>dialcodes</i> (F)
cref	<i>cref</i> (CP)	dialers	<i>dialers</i> (F)
cron	<i>cron</i> (C)	diecetomsbin	
crontab	<i>crontab</i> (C)	<i>diecetomsbin</i> (DOS)
crontab	<i>crontab</i> (C)	diff	<i>diff</i> (C)
crypt	<i>crypt</i> (C)	diff3	<i>diff3</i> (C)
crypt	<i>crypt</i> (S)	difftime	<i>difftime</i> (DOS)
cscanf	<i>cscanf</i> (DOS)	difftime	<i>difftime</i> (S)
cscope	<i>cscope</i> (CP)	dir	<i>dir</i> (F)
csh	<i>csh</i> (C)	dircmp	<i>dircmp</i> (C)
csplit	<i>csplit</i> (C)	dirent	<i>dirent</i> (F)
ct	<i>ct</i> (C)	dirname	<i>dirname</i> (C)
ctags	<i>ctags</i> (CP)	dis	<i>dis</i> (CP)
ctermid	<i>ctermid</i> (S)	disable	<i>disable</i> (C)
ctime	<i>ctime</i> (S)	discr	<i>discr</i> (S)
ctrace	<i>ctrace</i> (CP)	diskcmp	<i>diskcp</i> (C)
ctype	<i>ctype</i> (S)	diskcp	<i>diskcp</i> (C)
cu	<i>cu</i> (C)	diskusg	<i>diskusg</i> (ADM)
curses	<i>curses</i> (S)	displaypkg	
cuserid	<i>cuserid</i> (S)	<i>displaypkg</i> (ADM)
custom	<i>custom</i> (ADM)	div	<i>div</i> (DOS)
cut	<i>cut</i> (C)	div	<i>div</i> (S)
cvtcoff	<i>cvtcoff</i> (M)	divvy	<i>divvy</i> (ADM)
cvtomf	<i>cvtomf</i> (M)	dlvr_audit ..	<i>dlvr_audit</i> (ADM)
cxref	<i>cxref</i> (CP)	dmesg	<i>dmesg</i> (ADM)
daemon.mn	<i>daemon.mn</i> (M)	dmsbintoieee	
date	<i>date</i> (C)	<i>diecetomsbin</i> (DOS)
dblock	<i>dblock</i> (S)	dodisk	<i>acctsh</i> (ADM)
dbmbuild	<i>dbmbuild</i> (ADM)	doscat	<i>dos</i> (C)
dbmunit	<i>dbm</i> (S)	doscp	<i>dos</i> (C)
dc	<i>dc</i> (C)	dosdir	<i>dos</i> (C)
dcopy	<i>dcopy</i> (ADM)	dosexterr	<i>dosexterr</i> (DOS)
dd	<i>dd</i> (C)	dosformat	<i>dos</i> (C)
deassign	<i>assign</i> (C)	dosld	<i>dosld</i> (CP)
default	<i>default</i> (F)	dosls	<i>dos</i> (C)
defopen	<i>defopen</i> (S)	dosmkdir	<i>dos</i> (C)
defread	<i>defopen</i> (S)	dosrm	<i>dos</i> (C)
delete	<i>dbm</i> (S)	dosrmdir	<i>dos</i> (C)
deliver	<i>deliver</i> (ADM)	dparam	<i>dparam</i> (ADM)
delta	<i>delta</i> (CP)	drand48	<i>drand48</i> (S)
devices	<i>devices</i> (F)	dtox	<i>dtox</i> (C)
devnm	<i>devnm</i> (C)	dtype	<i>dtype</i> (C)
df	<i>df</i> (C)	du	<i>du</i> (C)

dump	<i>dump</i> (CP)	execle	<i>exec</i> (S)
dup	<i>dup</i> (S)	execlp	<i>exec</i> (DOS)
dup2	<i>dup2</i> (S)	execlp	<i>exec</i> (S)
echo	<i>echo</i> (C)	execlpe	<i>exec</i> (DOS)
ecvt	<i>ecvt</i> (S)	execseg	<i>execseg</i> (S)
ed	<i>ed</i> (C)	execv	<i>exec</i> (DOS)
edata	<i>end</i> (S)	execv	<i>exec</i> (S)
edit	<i>ex</i> (C)	execve	<i>exec</i> (DOS)
egrep	<i>grep</i> (C)	execve	<i>exec</i> (S)
enable	<i>enable</i> (C)	execvp	<i>exec</i> (DOS)
end	<i>end</i> (S)	execvp	<i>exec</i> (S)
enddvagent	<i>getdvagent</i> (S)	execvpe	<i>exec</i> (DOS)
endgrent	<i>getgrent</i> (S)	_exit	<i>exit</i> (DOS)
endprcment	<i>getprcment</i> (S)	_exit	<i>exit</i> (S)
endprdfent	<i>getprdfent</i> (S)	exp	<i>exp</i> (S)
endprfient	<i>getprfient</i> (S)	_expand	<i>_expand</i> (DOS)
endprpwent	<i>getprpwent</i> (S)	expr	<i>expr</i> (C)
endprtcnt	<i>getprtcnt</i> (S)	fabs	<i>floor</i> (S)
endpwent	<i>getpwent</i> (S)	factor	<i>factor</i> (C)
endutent	<i>getut</i> (S)	false	<i>false</i> (C)
env	<i>env</i> (C)	fclose	<i>fclose</i> (DOS)
environ	<i>environ</i> (M)	fclose	<i>fclose</i> (S)
eof	<i>eof</i> (DOS)	fcloseall	<i>fclose</i> (DOS)
erand48	<i>drand48</i> (S)	fcntl	<i>fcntl</i> (M)
erf	<i>erf</i> (S)	fcntl	<i>fcntl</i> (S)
erfc	<i>erf</i> (S)	fcvt	<i>ecvt</i> (S)
errno	<i>perror</i> (S)	fd	<i>fd</i> (HW)
error	<i>error</i> (M)	fdisk	<i>fdisk</i> (ADM)
etext	<i>end</i> (S)	fdopen	<i>fopen</i> (S)
ev_block	<i>ev_block</i> (S)	fdswap	<i>fdswap</i> (ADM)
ev_close	<i>ev_close</i> (S)	feof	<i>ferror</i> (S)
ev_count	<i>ev_count</i> (S)	ferror	<i>ferror</i> (S)
ev_flush	<i>ev_flush</i> (S)	fetch	<i>dbm</i> (S)
ev_getdev	<i>ev_getdev</i> (S)	ff	<i>ff</i> (ADM)
ev_getemask	<i>ev_getemask</i> (S)	fflush	<i>fclose</i> (S)
ev_gindev	<i>ev_gindev</i> (S)	_ffree	<i>free</i> (DOS)
ev_init	<i>ev_init</i> (S)	fgetc	<i>fgetc</i> (DOS)
ev_pop	<i>ev_pop</i> (S)	fgetc	<i>getc</i> (S)
ev_read	<i>ev_read</i> (S)	fgetchar	<i>fgetc</i> (DOS)
ev_resume	<i>ev_resume</i> (S)	fgetgrent	<i>getgrent</i> (S)
ev_setemask	<i>ev_setemask</i> (S)	fgetpos	<i>fgetpos</i> (DOS)
ev_suspend	<i>ev_suspend</i> (S)	fgetpos	<i>fgetpos</i> (S)
ex	<i>ex</i> (C)	fgetpwent	<i>getpwent</i> (S)
execl	<i>exec</i> (DOS)	fgets	<i>gets</i> (S)
execl	<i>exec</i> (S)	fgrep	<i>grep</i> (C)
execle	<i>exec</i> (DOS)	_fheapchk	<i>_fheapchk</i> (DOS)

_fheapwalk	_freet	_freet (DOS)
..... fheapwalk (DOS)	freopen	fopen (S)
fieetomsbin	freopen	freopen (DOS)
..... fieetomsbin (DOS)	frexp	frexp (S)
field	fsave	fsave (ADM)
field (S)	fscanf	scanf (S)
fields	fsck	fsck (ADM)
fields (S)	fsdb	fsdb (ADM)
fieldtype	fseek	fseek (DOS)
fieldtype (S)	fseek	fseek (S)
file	fsetpos	fsetpos (DOS)
file (C)	fsetpos	fsetpos (S)
filehdr	fsname	fsname (ADM)
filehdr (F)	fspec	fspec (F)
filelength	fsphoto	fsphoto (ADM)
filelength (DOS)	fsstat	fsstat (ADM)
fileno	fstat	fstat (DOS)
fileno (S)	fstat	stat (S)
filesys	fstatfs	statfs (S)
filesys (F)	fstyp	fstyp (ADM)
filesystem	ftell	fseek (S)
filesystem (F)	ftell	ftell (DOS)
find	ftime	ftime (DOS)
find (C)	ftok	stdipc (S)
finger	ftw	ftw (S)
finger (C)	fwrite	fread (S)
firstkey	fwrite	fwrite (DOS)
firstkey (S)	fwtmp	fwtmp (ADM)
fixhdr	fxlist	xlist (S)
fixhdr (C)	gamma	gamma (S)
fixperm	gcvt	ecvt (S)
fixperm (ADM)	genc	genc (CP)
floor	get	get (CP)
floor (S)	getc	getc (S)
flushall	getch	getch (DOS)
flushall (DOS)	getchar	getc (S)
fmalloc	getche	getch (DOS)
fmalloc (DOS)	getclk	getclk (M)
fmod	getcwd	getcwd (S)
fmod (S)	getdents	getdents (S)
fmsbintoieee	getdvagent	getdvagent (S)
..... fieetomsbin (DOS)	getdvagent	getdvagent (S)
fmsize	getgid	getuid (S)
fmsize (DOS)	getenv	getenv (S)
fopen	geteuid	getuid (S)
fopen (DOS)	getgid	getuid (S)
fopen		
fopen (S)		
fork		
fork (S)		
form		
form (S)		
format		
format (C)		
fpgetmask		
fpgetmask (S)		
fpgetround		
fpgetround (S)		
fpgetsticky		
fpgetsticky (S)		
fpreset		
fpreset (DOS)		
fprintf		
fprintf (S)		
fpsetmask		
fpsetmask (S)		
fpsetround		
fpsetround (S)		
fpsetsticky		
fpsetsticky (S)		
fputc		
fputc (DOS)		
fputc		
fputc (S)		
fputchar		
fputchar (DOS)		
fputs		
fputs (S)		
fread		
fread (DOS)		
fread		
fread (S)		
free		
free (DOS)		
free		
free (S)		

getgrent getgrent(S)
 getgrgid getgrent(S)
 getgrnam getgrent(S)
 getgroups getgroups(S)
 gethz gethz(S)
 getlogin getlogin(S)
 getluid getluid(S)
 getmsg getmsg(S)
 getopt getopt(C)
 getopt getopt(S)
 getoptcv getopt(C)
 getopt getopt(C)
 getpass getpass(S)
 getpasswd getpasswd(S)
 getpgrp getpid(S)
 getpid getpid(DOS)
 getpid getpid(S)
 getppid getpid(S)
 getprcment getprcment(S)
 getprcmnam getprcment(S)
 getprdfent getprdfent(S)
 getprdfnam getprdfent(S)
 getprfient getprfient(S)
 getprfinam getprfient(S)
 getpriv getpriv(S)
 getprpwent getprpwent(S)
 getprpwnam getprpwent(S)
 getprpwuid getprpwent(S)
 getprtcent getprtcent(S)
 getprtcnam getprtcent(S)
 getpw getpw(S)
 getpwent getpwent(S)
 getpwnam getpwent(S)
 getpwuid getpwent(S)
 gets gets(C)
 gets gets(S)
 getseed seed(S)
 getty getty(M)
 gettydefs gettydefs(F)
 getuid getuid(S)
 getutent getut(S)
 getutid getut(S)
 getutline getut(S)
 getw getc(S)
 gmtime ctime(S)
 goodpw goodpw(ADM)

gps gps(F)
 graph graph(ADM)
 greek greek(C)
 grep grep(C)
 gr_idtoname .. pw_mapping(S)
 gr_nametoid .. pw_mapping(S)
 group group(F)
 grpcheck grpcheck(C)
 gsignal ssignal(S)
 halloc halloc(DOS)
 haltsys haltsys(ADM)
 hashcheck spell(C)
 hashmake spell(C)
 hcreate hsearch(S)
 hd hd(C)
 hd hd(HW)
 hdestroy hsearch(S)
 hdr hdr(CP)
 head head(C)
 hello hello(C)
 help help(CP)
 hfree hfree(DOS)
 hp hp(C)
 hsearch hsearch(S)
 hwconfig hwconfig(C)
 hypot hypot(S)
 i286emul i286emul(C)
 i286emul i286emul(CP)
 i386 machid(C)
 id id(ADM)
 id id(C)
 idbuild idbuild(ADM)
 idcheck idcheck(ADM)
 idconfig idbuild(ADM)
 identity identity(S)
 idinstall idinstall(ADM)
 idleout idleout(ADM)
 idmkenv idbuild(ADM)
 idmkinit idmkinit(ADM)
 idmknod idmknod(ADM)
 idmkunix idbuild(ADM)
 idscsi idbuild(ADM)
 idspace idspace(ADM)
 idtune idtune(ADM)
 idvidi idbuild(ADM)
 infocmp infocmp(ADM)

init	<i>init</i> (M)	kill	<i>kill</i> (C)
init.base	<i>inittab</i> (F)	kill	<i>kill</i> (S)
initcond	<i>initcond</i> (ADM)	killall	<i>killall</i> (ADM)
inittab	<i>inittab</i> (F)	kmem	<i>mem</i> (F)
inode	<i>inode</i> (F)	l	<i>l</i> (C)
inp	<i>inp</i> (DOS)	l3tol	<i>l3tol</i> (S)
inpw	<i>inp</i> (DOS)	l64a	<i>a64l</i> (S)
install	<i>install</i> (ADM)	labelit	<i>labelit</i> (ADM)
installpkg	<i>installpkg</i> (ADM)	labs	<i>labs</i> (DOS)
int86	<i>int86</i> (DOS)	labs	<i>labs</i> (S)
int86x	<i>int86x</i> (DOS)	langinfo	<i>langinfo</i> (F)
intdos	<i>intdos</i> (DOS)	last	<i>last</i> (C)
intdosx	<i>intdosx</i> (DOS)	lastlogin	<i>acctsh</i> (ADM)
integrity	<i>integrity</i> (ADM)	layers	<i>layers</i> (C)
ioctl	<i>ioctl</i> (S)	layers	<i>layers</i> (M)
ipcrm	<i>ipcrm</i> (ADM)	lc	<i>lc</i> (C)
ipcs	<i>ipcs</i> (ADM)	lcong48	<i>drand48</i> (S)
isalnum	<i>ctype</i> (S)	ld	<i>ld</i> (CP)
isalpha	<i>ctype</i> (S)	ld	<i>ld</i> (M)
isascii	<i>ctype</i> (S)	ld	<i>ld</i> (XNX)
isatty	<i>isatty</i> (DOS)	ldaclose	<i>ldclose</i> (S)
isatty	<i>ttyname</i> (S)	ldahread	<i>ldahread</i> (S)
iscntrl	<i>ctype</i> (S)	ldaopen	<i>ldopen</i> (S)
isdigit	<i>ctype</i> (S)	ldclose	<i>ldclose</i> (S)
isgraph	<i>ctype</i> (S)	ldexp	<i>frexp</i> (S)
islower	<i>ctype</i> (S)	ldfcn	<i>ldfcn</i> (F)
ismpx	<i>ismpx</i> (C)	ldfhread	<i>ldfhread</i> (S)
isnand	<i>isnan</i> (S)	ldgetname	<i>ldgetname</i> (S)
isnanf	<i>isnan</i> (S)	ldiv	<i>ldiv</i> (DOS)
isprint	<i>ctype</i> (S)	ldiv	<i>ldiv</i> (S)
ispunct	<i>ctype</i> (S)	ldlinit	<i>ldlread</i> (S)
isspace	<i>ctype</i> (S)	ldlitem	<i>ldlread</i> (S)
issue	<i>issue</i> (F)	ldlread	<i>ldlread</i> (S)
isupper	<i>ctype</i> (S)	ldlseek	<i>ldlseek</i> (S)
isxdigit	<i>ctype</i> (S)	ldlnseek	<i>ldlseek</i> (S)
item	<i>item</i> (S)	ldnrseek	<i>ldrseek</i> (S)
itoa	<i>itoa</i> (DOS)	ldnshread	<i>ldshread</i> (S)
j0, j1, jn	<i>bessel</i> (S)	ldnsseek	<i>ldsseek</i> (S)
jagent	<i>jagent</i> (M)	ldohseek	<i>ldohseek</i> (S)
join	<i>join</i> (C)	ldopen	<i>ldopen</i> (S)
jrand48	<i>drand48</i> (S)	ldrseek	<i>ldrseek</i> (S)
jterm	<i>jterm</i> (C)	ldshread	<i>ldshread</i> (S)
jwin	<i>jwin</i> (C)	ldsseek	<i>ldsseek</i> (S)
kbhit	<i>kbhit</i> (DOS)	ldtbindex	<i>ldtbindex</i> (S)
kbmode	<i>kbmode</i> (ADM)	ldtbread	<i>ldtbread</i> (S)
keyboard	<i>keyboard</i> (HW)	ldtbseek	<i>ldtbseek</i> (S)

lex	<i>lex</i> (CP)	lsearch	<i>lsearch</i> (S)
lfind	<i>lfind</i> (DOS)	lseek	<i>lseek</i> (DOS)
lfind	<i>lsearch</i> (S)	lseek	<i>lseek</i> (S)
libwindows	<i>libwindows</i> (S)	ltoa	<i>ltoa</i> (DOS)
limits	<i>limits</i> (F)	ltol3	<i>l3tol</i> (S)
line	<i>line</i> (C)	m4	<i>m4</i> (CP)
linenum	<i>linenum</i> (F)	machine	<i>machine</i> (HW)
link	<i>link</i> (ADM)	mail	<i>mail</i> (C)
link	<i>link</i> (S)	majorsinuse	<i>majorsinuse</i> (ADM)
link_unix	<i>link_unix</i> (ADM)	make	<i>make</i> (CP)
lint	<i>lint</i> (CP)	makekey	<i>makekey</i> (ADM)
list	<i>list</i> (ADM)	malloc	<i>malloc</i> (DOS)
list	<i>list</i> (CP)	malloc	<i>malloc</i> (S)
ln	<i>ln</i> (C)	man	<i>man</i> (C)
locale	<i>locale</i> (M)	mapchan	<i>mapchan</i> (F)
localtime	<i>ctime</i> (S)	mapchan	<i>mapchan</i> (M)
lock	<i>lock</i> (C)	mapkey	<i>mapkey</i> (M)
lock	<i>lock</i> (S)	mapscrn	<i>mapkey</i> (M)
lockf	<i>lockf</i> (S)	mapstr	<i>mapkey</i> (M)
locking	<i>locking</i> (DOS)	masm	<i>masm</i> (CP)
locking	<i>locking</i> (S)	masm	<i>masm</i> (CP)
log	<i>exp</i> (S)	math	<i>math</i> (M)
log	<i>log</i> (STR)	matherr	<i>matherr</i> (S)
log10	<i>exp</i> (S)	maxuuscheds	<i>maxuuscheds</i> (F)
login	<i>login</i> (M)	maxuuxqts	<i>maxuuxqts</i> (F)
logname	<i>logname</i> (C)	mcs	<i>mcs</i> (CP)
logname	<i>logname</i> (S)	mdevice	<i>mdevice</i> (F)
logs	<i>logs</i> (F)	mem	<i>mem</i> (F)
longjmp	<i>setjmp</i> (S)	_memavl	<i>_memavl</i> (DOS)
lorder	<i>lorder</i> (CP)	memccpy	<i>memory</i> (S)
lp: lp, lp0, lp1, lp2	<i>lp</i> (HW)	memchr	<i>memory</i> (S)
lpadmin	<i>lpadmin</i> (ADM)	memcmp	<i>memory</i> (S)
lpfilter	<i>lpfilter</i> (ADM)	memcpy	<i>memory</i> (S)
lpforms	<i>lpforms</i> (ADM)	memicmp	<i>memicmp</i> (DOS)
lpmove	<i>lpsched</i> (ADM)	memmove	<i>memmove</i> (DOS)
lprint	<i>lprint</i> (C)	memmove	<i>memmove</i> (S)
lprof	<i>lprof</i> (CP)	memset	<i>memory</i> (S)
lpsched	<i>lpsched</i> (ADM)	menu	<i>menu</i> (S)
lpsh	<i>lpsh</i> (ADM)	mesg	<i>mesg</i> (C)
lpshut	<i>lpsched</i> (ADM)	messages	<i>messages</i> (M)
lpstat	<i>lpstat</i> (C)	mestbl	<i>mestbl</i> (M)
lpusers	<i>lpusers</i> (ADM)	mfsys	<i>mfsys</i> (F)
lrand48	<i>drand48</i> (S)	micnet	<i>micnet</i> (F)
ls	<i>ls</i> (C)	mkdev	<i>mkdev</i> (ADM)
lsearch	<i>lfind</i> (DOS)		

mkdir	<i>mkdir</i> (C)	news	<i>news</i> (C)
mkdir	<i>mkdir</i> (DOS)	nextkey	<i>dbm</i> (S)
mkdir	<i>mkdir</i> (S)	_nfree	<i>free</i> (DOS)
mkfifo	<i>mkfifo</i> (S)	_nheapchk ...	<i>fheapchk</i> (DOS)
mkfs	<i>mkfs</i> (ADM)	_nheapwalk
mknod	<i>mknod</i> (C)	<i>fheapwalk</i> (DOS)
mknod	<i>mknod</i> (S)	nice	<i>nice</i> (C)
mkshlib	<i>mkshlib</i> (CP)	nice	<i>nice</i> (S)
mkstr	<i>mkstr</i> (CP)	nictable	<i>nictable</i> (ADM)
mkstr	<i>mkstr</i> (CP)	nl	<i>nl</i> (C)
mktemp	<i>mktemp</i> (S)	nl_ascxtime	<i>nl_cxtime</i> (S)
mktime	<i>mktime</i> (S)	nl_cxtime	<i>nl_cxtime</i> (S)
mmdf	<i>mmdf</i> (ADM)	nl_fprintf	<i>nl_printf</i> (S)
mmdfalias ..	<i>mmdfalias</i> (ADM)	nl_fscanf	<i>nl_scanf</i> (S)
mmdftailor	<i>mmdftailor</i> (F)	nl_init	<i>nl_init</i> (S)
mnlist	<i>mnlist</i> (ADM)	nlist	<i>nlist</i> (S)
mnt	<i>mnt</i> (C)	nl_langinfo	<i>nl_langinfo</i> (S)
mnttab	<i>mnttab</i> (F)	nl_printf	<i>nl_printf</i> (S)
modf	<i>frexp</i> (S)	nlscanf	<i>nlscanf</i> (ADM)
monacct	<i>acctsh</i> (ADM)	nl_scanf	<i>nl_scanf</i> (S)
monitor	<i>monitor</i> (S)	nl_sprintf	<i>nl_printf</i> (S)
montbl	<i>montbl</i> (M)	nl_sscanf	<i>nl_sscanf</i> (S)
more	<i>more</i> (C)	nl_strcmp	<i>nl_strcmp</i> (S)
mount	<i>mount</i> (ADM)	nl_strncmp	<i>nl_strcmp</i> (S)
mount	<i>mount</i> (S)	nl_types	<i>nl_types</i> (F)
mountall	<i>mountall</i> (ADM)	nm	<i>nm</i> (CP)
mouse	<i>mouse</i> (HW)	nm	<i>nm</i> (XNX)
movedata	<i>movedata</i> (DOS)	_nmalloc	<i>malloc</i> (DOS)
mrnd48	<i>drand48</i> (S)	_nmsize	<i>_msize</i> (DOS)
mscreen	<i>mscreen</i> (M)	nohup	<i>nohup</i> (C)
msgctl	<i>msgctl</i> (S)	nopromain	<i>promain</i> (M)
msgget	<i>msgget</i> (S)	nrnd48	<i>drand48</i> (S)
msgrcv	<i>msgop</i> (S)	null	<i>null</i> (F)
msgsnd	<i>msgop</i> (S)	nulladm	<i>acctsh</i> (ADM)
_msize	<i>_msize</i> (DOS)	numtbl	<i>numtbl</i> (M)
mtune	<i>mtune</i> (F)	od	<i>od</i> (C)
multiscreen	<i>multiscreen</i> (M)	open	<i>open</i> (DOS)
mv	<i>mv</i> (C)	open	<i>open</i> (S)
mvdevice	<i>mvdevice</i> (F)	opendir	<i>directory</i> (S)
mvdir	<i>mvdir</i> (ADM)	opensem	<i>opensem</i> (S)
nap	<i>nap</i> (S)	os2ld	<i>os2ld</i> (CP)
nbwaitsem	<i>waitsem</i> (S)	outp	<i>outp</i> (DOS)
ncheck	<i>ncheck</i> (ADM)	outpw	<i>outp</i> (DOS)
netutil	<i>netutil</i> (ADM)	paccess	<i>paccess</i> (S)
newform	<i>newform</i> (C)	pack	<i>pack</i> (C)
newgrp	<i>newgrp</i> (C)	panel	<i>panel</i> (S)

parallel	<i>parallel</i> (HW)	ptrace	<i>ptrace</i> (S)
passlen	<i>passlen</i> (S)	purge	<i>purge</i> (C)
passwd	<i>passwd</i> (C)	purge	<i>purge</i> (F)
passwd	<i>passwd</i> (F)	putc	<i>putc</i> (S)
paste	<i>paste</i> (C)	putch	<i>putch</i> (DOS)
pathconf	<i>pathconf</i> (S)	putchar	<i>putc</i> (S)
pause	<i>pause</i> (S)	putdvagnam	<i>getdvagent</i> (S)
pax	<i>pax</i> (C)	putenv	<i>putenv</i> (S)
pcat	<i>pack</i> (C)	putmsg	<i>putmsg</i> (S)
pclose	<i>popen</i> (S)	putprcmnam	<i>getprcment</i> (S)
pcpio	<i>pcpio</i> (C)	putprdfnam	<i>getprdfent</i> (S)
permissions	<i>permissions</i> (F)	putprfinam	<i>getprfient</i> (S)
perror	<i>perror</i> (S)	putprpwnam	<i>getprpwent</i> (S)
pg	<i>pg</i> (C)	putprtcnam	<i>getprtcent</i> (S)
pipe	<i>pipe</i> (S)	putpwent	<i>putpwent</i> (S)
plock	<i>plock</i> (S)	puts	<i>puts</i> (S)
plot	<i>plot</i> (F)	pututline	<i>getut</i> (S)
plot	<i>plot</i> (S)	putw	<i>putc</i> (S)
pnch	<i>pnch</i> (F)	pwcheck	<i>pwcheck</i> (C)
poll	<i>poll</i> (F)	pwd	<i>pwd</i> (C)
poll	<i>poll</i> (S)	pw_idtoname	
popen	<i>popen</i> (S)	<i>pw_mapping</i> (S)
pow	<i>exp</i> (S)	pw_nametoid	
pr	<i>pr</i> (C)	<i>pw_mapping</i> (S)
prctmp	<i>acctsh</i> (ADM)	qsort	<i>qsort</i> (S)
prdaily	<i>acctsh</i> (ADM)	queue	<i>queue</i> (F)
prf	<i>prf</i> (HW)	queuedefs	<i>queuedefs</i> (F)
prfdc	<i>profiler</i> (ADM)	quot	<i>quot</i> (C)
prfld	<i>profiler</i> (ADM)	raise	<i>raise</i> (DOS)
prfpr	<i>profiler</i> (ADM)	raise	<i>raise</i> (S)
prfsnap	<i>profiler</i> (ADM)	ramdisk	<i>ramdisk</i> (HW)
prfstat	<i>profiler</i> (ADM)	rand	<i>rand</i> (S)
printf	<i>printf</i> (S)	random	<i>random</i> (C)
proctl	<i>proctl</i> (S)	randomword	
prof	<i>prof</i> (CP)	<i>randomword</i> (S)
prof	<i>prof</i> (M)	ranlib	<i>ranlib</i> (CP)
prof	<i>prof</i> (XNX)	ranlib	<i>ranlib</i> (CP)
profil	<i>profil</i> (S)	rc0	<i>rc0</i> (ADM)
profile	<i>profile</i> (M)	rc2	<i>rc2</i> (ADM)
promain	<i>promain</i> (M)	rcc	<i>rcc</i> (CP)
proto	<i>proto</i> (ADM)	rcp	<i>rcp</i> (C)
prs	<i>prs</i> (CP)	rdchk	<i>rdchk</i> (S)
prtacct	<i>acctsh</i> (ADM)	read	<i>read</i> (DOS)
ps	<i>ps</i> (C)	read	<i>read</i> (S)
pstat	<i>pstat</i> (C)	readdir	<i>directory</i> (S)
ptar	<i>ptar</i> (C)	realloc	<i>malloc</i> (S)

reboot	haltsys (ADM)	scnhdr	scnhdr (F)
red	ed (C)	scr_dump	scr_dump (F)
reduce	reduce (ADM)	screen	screen (HW)
regcmp	regcmp (CP)	scsi	scsi (HW)
regcmp	regcmp (S)	sdb	sdb (CP)
regcmp	regex (S)	sddate	sddate (C)
regex	regcmp (S)	sdenter	sdenter (S)
regex	regex (S)	sdevice	sdevice (F)
regexp	regexp (S)	sdfree	sdget (S)
reject	accept (ADM)	sdget	sdget (S)
reloc	reloc (F)	sdgetv	sdgetv (S)
relogin	relogin (ADM)	sdiff	sdiff (C)
remote	remote (C)	sdleave	sdenter (S)
remove	remove (DOS)	sdwaitv	sdgetv (S)
remove	remove (S)	sed	sed (C)
removepkg		seed48	drand48 (S)
.....	removepkg (ADM)	seekdir	directory (S)
rename	rename (DOS)	segread	segread (DOS)
rename	rename (S)	select	select (S)
restore	restore (ADM)	semctl	semctl (S)
rewind	fseek (S)	semget	semget (S)
rewinddir	directory (S)	semop	semop (S)
rm	rm (C)	serial	serial (HW)
rmail	rmail (ADM)	setbuf	setbuf (S)
rmb	rmb (M)	setclock	setclock (ADM)
rmdel	rmdel (CP)	setcolor	setcolor (C)
rmdir	rmdir (C)	setcolour	setcolor (C)
rmdir	rmdir (DOS)	setdvagent	getdvagent (S)
rmdir	rmdir (S)	setgid	setuid (S)
rmtmp	rmtmp (DOS)	setgrent	getgrent (S)
routines	routines (ADM)	setgroups	setgroups (S)
rsh	rsh (C)	setjmp	setjmp (S)
rtc	rtc (HW)	setkey	setkey (C)
runacct	acctsh (ADM)	setlocale	setlocale (S)
runacct	runacct (ADM)	setluid	setluid (S)
sa1	sar (ADM)	setmnt	setmnt (ADM)
sa2	sar (ADM)	setmode	setmode (DOS)
sact	sact (CP)	setpgid	setpgid (S)
sadc	sar (ADM)	setpgrp	setpgrp (S)
sag	sag (ADM)	setprcmnt	getprcmnt (S)
sar	sar (ADM)	setprdfent	getprdfent (S)
sbrk	brk (S)	setprfient	getprfient (S)
scanf	scanf (S)	setpriv	setpriv (S)
scsdiff	scsdiff (CP)	setprpwent	getprpwent (S)
scsfile	scsfile (F)	setprtcent	getprtcent (S)
schedule	schedule (ADM)	setpwent	getpwent (S)

setseed *seed*(S)
 setsid *setsid*(S)
 settime *settime*(ADM)
 setuid *setuid*(S)
 setutent *getut*(S)
 setvbuf *setbuf*(S)
 setvbuf *setvbuf*(DOS)
 sfmt *sfmt*(ADM)
 sfsys *sfsys*(F)
 sgetl *spu*l(S)
 sh *sh*(C)
 shl *shl*(C)
 shmat *shmop*(S)
 shmctl *shmctl*(S)
 shmdt *shmop*(S)
 shmget *shmget*(S)
 shutacct *acctsh*(ADM)
 shutdown *shutdown*(S)
 shutdown *shutdown*(ADM)
 sigaction *sigaction*(S)
 siglongjmp *sigsetjmp*(S)
 signal *signal*(DOS)
 signal *signal*(S)
 sigpending *sigpending*(S)
 sigprocmask .. *sigprocmask*(S)
 sigsem *sigsem*(S)
 sigset *sigset*(S)
 sigsetjmp *sigsetjmp*(S)
 sigsuspend *sigsuspend*(S)
 sin *trig*(S)
 sinh *sinh*(S)
 size *size*(CP)
 size *size*(XNX)
 sleep *sleep*(C)
 sleep *sleep*(S)
 sopen *sopen*(DOS)
 sort *sort*(C)
 spawnl *spawn*(DOS)
 spawnle *spawn*(DOS)
 spawnlp *spawn*(DOS)
 spawnlpe *spawn*(DOS)
 spawnv *spawn*(DOS)
 spawnve *spawn*(DOS)
 spawnvp *spawn*(DOS)
 spawnvpe *spawn*(DOS)
 spell *spell*(C)

spellin *spell*(C)
 spline *spline*(C)
 split *split*(C)
 sprintf *printf*(S)
 spu^l *spu*l(S)
 sqrt *exp*(S)
 srand *rand*(S)
 srand48 *drand48*(S)
 sscanf *scanf*(S)
 ssignal *ssignal*(S)
 stackavail *stackavail*(DOS)
 startup *acctsh*(ADM)
 stat *stat*(DOS)
 stat *stat*(F)
 stat *stat*(S)
 statfs *statfs*(S)
 _status87 *_status87*(DOS)
 stdio *stdio*(S)
 stime *stime*(S)
 stopio *stopio*(S)
 store *dbm*(S)
 strace *strace*(ADM)
 strcat *strcat*(DOS)
 strcat *string*(S)
 strchr *strcat*(DOS)
 strchr *string*(S)
 strclean *strclean*(ADM)
 strcmp *string*(S)
 strcmpi *strcat*(DOS)
 strcoll *collation*(S)
 strcpy *strcat*(DOS)
 strcpy *string*(S)
 strcspn *strcat*(DOS)
 strcspn *string*(S)
 strdup *strcat*(DOS)
 strdup *string*(S)
 streamio *streamio*(STR)
 strerr *strerr*(ADM)
 strerror *strerror*(DOS)
 strerror *strerror*(S)
 strftime *ctime*(S)
 strftime *strftime*(S)
 stricmp *strcat*(DOS)
 strings *strings*(C)
 strip *strip*(CP)
 strip *strip*(XNX)

strlen	<i>string</i> (S)	system	<i>system</i> (DOS)
strlwr	<i>strlwr</i> (DOS)	system	<i>system</i> (S)
strncat	<i>string</i> (S)	systemid	<i>systemid</i> (F)
strncat	<i>strncat</i> (DOS)	systems	<i>systems</i> (F)
strncmp	<i>string</i> (S)	systty	<i>systty</i> (M)
strncmp	<i>strncat</i> (DOS)	tables	<i>tables</i> (F)
strncoll	<i>collation</i> (S)	tabs	<i>tabs</i> (C)
strncpy	<i>string</i> (S)	t_accept	<i>t_accept</i> (NSL)
strncpy	<i>strncat</i> (DOS)	tail	<i>tail</i> (C)
strnicmp	<i>strncat</i> (DOS)	t_alloc	<i>t_accept</i> (NSL)
strnset	<i>strncat</i> (DOS)	t_alloc	<i>t_alloc</i> (NSL)
strnxfm	<i>collation</i> (S)	tam	<i>tam</i> (S)
strpbrk	<i>string</i> (S)	tan	<i>trig</i> (S)
strrchr	<i>string</i> (S)	tanh	<i>sinh</i> (S)
strrev	<i>strrev</i> (DOS)	tape	<i>tape</i> (C)
strset	<i>strset</i> (DOS)	tape	<i>tape</i> (HW)
strspn	<i>string</i> (S)	tapecntl	<i>tapecntl</i> (C)
strstr	<i>strstr</i> (DOS)	tapedump	<i>tapedump</i> (C)
strtod	<i>strtod</i> (S)	tar	<i>tar</i> (C)
strtok	<i>string</i> (S)	tar	<i>tar</i> (F)
strtol	<i>strtol</i> (S)	t_bind	<i>t_accept</i> (NSL)
strupr	<i>strupr</i> (DOS)	t_bind	<i>t_bind</i> (NSL)
strxfm	<i>collation</i> (S)	tcdrain	<i>tcflow</i> (S)
stty	<i>stty</i> (C)	tcflow	<i>tcflow</i> (S)
stune	<i>stune</i> (F)	tcflush	<i>tcflow</i> (S)
su	<i>su</i> (C)	tcgetattr	<i>tcattr</i> (S)
submit	<i>submit</i> (ADM)	tcgetpgrp	<i>tcpgrp</i> (S)
subsystem	<i>subsystem</i> (M)	t_close	<i>t_accept</i> (NSL)
subsystems	<i>subsystems</i> (S)	t_close	<i>t_close</i> (NSL)
sulogin	<i>sulogin</i> (ADM)	t_connect	<i>t_accept</i> (NSL)
sum	<i>sum</i> (C)	t_connect	<i>t_connect</i> (NSL)
swab	<i>swab</i> (S)	tcsendbreak	<i>tcflow</i> (S)
swap	<i>swap</i> (ADM)	tcsetattr	<i>tcattr</i> (S)
swconfig	<i>swconfig</i> (C)	tcsetpgrp	<i>tcpgrp</i> (S)
sxt	<i>sxt</i> (M)	tdelete	<i>tsearch</i> (S)
syms	<i>syms</i> (F)	tee	<i>tee</i> (C)
sync	<i>sync</i> (ADM)	telinit	<i>init</i> (M)
sync	<i>sync</i> (S)	tell	<i>tell</i> (DOS)
sysadmsh	<i>sysadmsh</i> (ADM)	telldir	<i>directory</i> (S)
sysconf	<i>sysconf</i> (S)	tempnam	<i>tmpnam</i> (S)
sysdef	<i>sysdef</i> (ADM)	term	<i>term</i> (F)
sys_errlist	<i>perror</i> (S)	term	<i>term</i> (M)
sysfiles	<i>sysfiles</i> (F)	termcap	<i>termcap</i> (F)
sysfs	<i>sysfs</i> (S)	terminal	<i>terminal</i> (HW)
sysi86	<i>sysi86</i> (S)	terminals	<i>terminals</i> (M)
sys_nerr	<i>perror</i> (S)	terminfo	<i>terminfo</i> (F)

terminfo terminfo (M)
 terminfo terminfo (S)
 termio termio (M)
 termios termios (M)
 t_error t_accept (NSL)
 t_error t_error (NSL)
 test test (C)
 tfind tsearch (S)
 t_free t_accept (NSL)
 t_free t_free (NSL)
 tgetent termcap (S)
 tgetflag termcap (S)
 t_getinfo t_accept (NSL)
 t_getinfo t_getinfo (NSL)
 tgetnum termcap (S)
 t_getstate t_accept (NSL)
 t_getstate t_getstate (NSL)
 tgetstr termcap (S)
 tgoto termcap (S)
 tic tic (C)
 time time (C)
 time time (S)
 times times (S)
 timex timex (ADM)
 timezone timezone (F)
 timod timod (STR)
 timtbl timtbl (M)
 tirdwr tirdwr (STR)
 t_listen t_accept (NSL)
 t_listen t_listen (NSL)
 t_look t_accept (NSL)
 t_look t_look (NSL)
 tmpfile tmpfile (S)
 tmpnam tmpnam (S)
 toascii conv (S)
 todigit conv (S)
 toint conv (S)
 tolower conv (S)
 top top (F)
 t_open t_accept (NSL)
 t_open t_open (NSL)
 top.next top (F)
 t_optmgmt t_accept (NSL)
 t_optmgmt t_optmgmt (NSL)
 touch touch (C)
 toupper conv (S)

tplot tplot (ADM)
 tput tput (C)
 tputs termcap (S)
 tr tr (C)
 translate translate (C)
 trchan trchan (M)
 t_rcv t_accept (NSL)
 t_rcv t_rcv (NSL)
 t_rcvconnect ... t_accept (NSL)
 t_rcvconnect
 t_rcvconnect (NSL)
 t_rcvdis t_accept (NSL)
 t_rcvdis t_rcvdis (NSL)
 t_rcvrel t_accept (NSL)
 t_rcvrel t_rcvrel (NSL)
 t_rcvudata t_accept (NSL)
 t_rcvudata ... t_rcvudata (NSL)
 t_rcvuderr t_accept (NSL)
 t_rcvuderr ... t_rcvuderr (NSL)
 true true (C)
 tsearch tsearch (S)
 tset tset (C)
 t_snd t_accept (NSL)
 t_snd t_snd (NSL)
 t_snddis t_accept (NSL)
 t_snddis t_snddis (NSL)
 t_sndrel t_accept (NSL)
 t_sndrel t_sndrel (NSL)
 t_sndudata t_accept (NSL)
 t_sndudata .. t_sndudata (NSL)
 tsort tsort (CP)
 t_sync t_accept (NSL)
 t_sync t_sync (NSL)
 tty tty (C)
 tty tty (M)
 tty1[A-H] serial (HW)
 tty2[A-H] serial (HW)
 tty2[a-h] serial (HW)
 ttyname ttyname (S)
 ttyslot ttyslot (S)
 t_unbind t_accept (NSL)
 t_unbind t_unbind (NSL)
 turnacct acctsh (ADM)
 twalk tsearch (S)
 types types (F)
 tz tz (M)

tzset	ctime(S)	uustat	uustat(C)
uadmin	uadmin(ADM)	uuto	uuto(C)
uadmin	uadmin(S)	uutry	uutry(ADM)
ulimit	ulimit(S)	uux	uux(C)
ultoa	ultoa(DOS)	uuxqt	uuxqt(ADM)
umask	umask(C)	val	val(CP)
umask	umask(DOS)	values	values(M)
umask	umask(S)	varargs	varargs(S)
umnt	mnt(C)	vc	vc(C)
umount	umount(ADM)	vc	vc(CP)
umount	umount(S)	vectorsinuse	
umountall	mountall(ADM)	vectorsinuse(ADM)
uname	uname(C)	vedit	vi(C)
uname	uname(S)	vfprintf	vprintf(S)
uncompress	compress(C)	vi	vi(C)
unget	unget(CP)	vidi	vidi(C)
ungetc	ungetc(S)	view	vi(C)
ungetch	ungetch(DOS)	vmstat	vmstat(C)
uniq	uniq(C)	volcopy	volcopy(ADM)
unistd	unistd(F)	vprintf	vprintf(S)
units	units(C)	vsprintf	vprintf(S)
unlink	link(ADM)	w	w(C)
unlink	unlink(DOS)	wait	wait(C)
unlink	unlink(S)	wait	wait(S)
unpack	pack(C)	waitpid	wait(S)
uptime	uptime(C)	waitsem	waitsem(S)
usemouse	usemouse(C)	wall	wall(ADM)
ustat	ustat(S)	wc	wc(C)
utime	utime(DOS)	what	what(C)
utime	utime(S)	what	what(CP)
utmp	utmp(F)	who	who(C)
utmpname	getut(S)	whodo	whodo(C)
uuchat	dial(ADM)	write	write(C)
uuchek	uuchek(ADM)	write	write(DOS)
uucico	uucico(ADM)	write	write(S)
uuclean	uuclean(ADM)	wtini	wtini(ADM)
uucp	uucp(C)	wtmp	utmp(F)
uudecode	uencode(C)	wtmpfix	fwtmp(ADM)
uencode	uencode(C)	x286emul	x286emul(C)
uugetty	uugetty(ADM)	x286emul	x286emul(CP)
uuinstall	uuinstall(ADM)	xargs	xargs(C)
uulist	uulist(ADM)	xbackup	xbackup(ADM)
uulog	uucp(C)	xbackup	xbackup(F)
uuname	uucp(C)	xdumpdir	xdumpdir(ADM)
uupick	uuto(C)	xinstall	xinstall(ADM)
uusched	uusched(ADM)	xlist	xlist(S)

x.out *x.out*(F)
xref *xref*(CP)
xrestor *xrestore*(ADM)
xrestore *xrestore*(ADM)
xstr *xstr*(CP)
xt *xt*(HW)
xtd *xtd*(ADM)
xtod *xtod*(C)
xtproto *xtproto*(M)
xts *xts*(ADM)
xtt *xtt*(ADM)
y0, y1, yn *bessel*(S)
yacc *yacc*(CP)
yes *yes*(C)
zcat *compress*(C)

Contents

Programming Commands (CP)

Intro	introduces Development System commands
adb	invokes a general-purpose debugger.
admin	create and administer SCCS files
ar	archive and library maintainer for portable archives
as	common assembler
cb	C program beautifier
cc	invokes the C compiler.
cdc	change the delta commentary of an SCCS delta
cflow	generate C flowgraph
chkshlib	compare shared libraries tool
codeview	visual debugger
comb	combine SCCS deltas
conv	common object file converter
convert	convert archive files to common formats
cpp	the C language preprocessor
cprs	compress a common object file
cref	makes a cross-reference listing
cscope	interactively examine a C program
ctags	create a tags file
ctrace	C program debugger
cxref	generate C program cross-reference
delta	make a delta (change) to an SCCS file
dis	object code disassembler
dosld	MS-DOS cross linker
dump	dump selected parts of an object file
gencc	create a front-end to the cc command
get	get a version of an SCCS file
hdr	display selected parts of an object file
help	asks for help about SCCS commands
i286emul	emulate 80286
ld	invokes the link editor
lex	generate programs for simple lexical tasks
lint	a C program checker
list	produce C source listing from a common object file
lorder	find ordering relation for an object library
lprof	display line-by-line execution count profile data
m4	macroprocessor

make	maintain, update, and regenerate groups of programs
masm	invokes the assembler
mcs	manipulate the object file comment section
mkshlib	create a shared library
mkstr	creates an error message file from C source
nm	print name list of common object file
os2ld	OS/2 cross linker
prof	display profile data
prs	print an SCCS file
ranlib	converts archives to random libraries.
rcc	AT&T C compiler
regcmp	regular expression compile
rmdel	remove a delta from an SCCS file
sact	print current SCCS file editing activity
sccsdiff	compare two versions of an SCCS file
sdb	symbolic debugger
size	print section sizes in bytes of common object files
strip	strip symbol and line number information from a common object file
tsort	topological sort
unget	undo a previous get of an SCCS file
val	validate SCCS file
vc	version control
what	identify SCCS files
x286emul	emulate XENIX 80286
xref	cross-references C programs
xstr	extracts strings from C programs
yacc	yet another compiler-compiler

Intro

introduces Development System commands

Description

This section describes use of the individual standalone commands available in the Development System. Each individual command is labeled with the letters CP to distinguish it from commands available in the Operating System and other commands within the Development System. These letters are used for easy reference from other documentation. For example, the reference *cc*(CP) indicates a reference to a discussion of the *cc* command in this section, where the letter "C" stands for "Command" and the letter "P" stands for "Programming".

Syntax

Unless otherwise noted, commands described in this section accept options and other arguments according to the following syntax:

name [*options*] [*cmdarg*]

where:

name The filename or pathname of an executable file

option A single letter representing a command option. By convention, most options are preceded with a dash. Option letters can sometimes be grouped together as in *-abcd* or alternatively they are specified individually as in *-a -b -c -d*. The method of specifying options depends on the syntax of the individual command. In the latter method of specifying options, arguments can be given to the options. For example, the *-f* option for many commands often takes a following filename argument.

cmdarg A pathname or other command argument *not* beginning with a dash. It may also be a dash alone by itself indicating the standard input.

See Also

getopt(C), getopt(S)

Diagnostics

Upon termination, each command returns 2 bytes of status, one supplied by the system and giving the cause for termination, and (in the case of "normal" termination) one supplied by the program (see *wait(S)* and *exit(S)*). The former byte is 0 for normal termination; the latter is customarily 0 for successful execution and nonzero to indicate troubles such as erroneous parameters, or bad or inaccessible data. It is called variously "exit code", "exit status", or "return code", and is described only where special conventions are involved.

Notes

Not all commands adhere to the above syntax.

adb

invokes a general-purpose debugger.

Syntax

adb [-w] [-p prompt] [objfil [corefile]

Description

adb is a general purpose debugging program. It may be used to examine files and to provide a controlled environment for the execution of binaries.

objfil is normally an executable program file of either UNIX format or COFF, preferably containing a symbol table; if not then the symbolic features of *adb* cannot be used although the file can still be examined. The default filename for *objfil* is **a.out**. *corefile* is assumed to be a core image file produced after executing *objfil*; the default for *corefile* is **core**.

Requests to *adb* are read from the standard input and responses are to the standard output. If the -w option is present then both *objfil* and *corefile* are created if necessary and opened for reading and writing so that files can be modified using *adb*. The QUIT and INTERRUPT keys cause *adb* to return to the next command. The -p option defines the prompt string. It may be any combination of characters. The default is an asterisk (*).

In general requests to *adb* are of the form:

[*address*] [, *count*] [*command*] [;]

If *address* is present then the current position in the file (called *dot*) is set to the value given by *address*. Initially *dot* is set to 0. For most commands *count* specifies how many times the command will be executed. The default *count* is 1. *address* is a special expression having the form:

[*segment*:]*offset*

where *segment* gives the address of a specific text or data segment, and *offset* gives an offset from the beginning of that segment. If *segment* is not given, the last segment value given in a command is used.

The interpretation of an address depends on the context it is used in. If a subprocess is being debugged then addresses are interpreted in the usual way in the address space of the subprocess. For further details of address mapping see *Addresses*.

Expressions

- The value of *dot*.
- ++ The value of *dot* incremented by the current increment.
- The value of *dot* decremented by the current increment.
- " The last *address* typed.
- integer* An octal number if *integer* begins with a 0; a hexadecimal number if preceded by # or 0x; otherwise a decimal number.
- integer fraction* A 32-bit floating point number.
- '*cccc*' The ASCII value of up to 4 characters. \ may be used to escape a '.
- < *name* The value of *name*, which is either a variable name or a register name. *adb* maintains a number of variables (see *Variables*) named by single letters or digits. If *name* is a register name then the value of the register is obtained from the system header in *corefile*. The register names are **ax bx cx dx di si bp fi ip cs ds ss es sp**. The name **fi** refers to the status flags.
- symbol* A *symbol* is a sequence of upper or lower case letters, underscores or digits, not starting with a digit. The value of the *symbol* is taken from the symbol table in *objfil*. An initial _ or ~ will be prepended to *symbol* if needed.
- _ *symbol* In C, the 'true name' of an external symbol begins with _. It may be necessary to use this name to distinguish it from internal or hidden variables of a program.
- (*exp*) The value of the expression *exp*.

Monadic operators

- **exp* The contents of the location addressed by *exp*.
- exp* Integer negation.
- ~*exp* Bitwise complement.

Dyadic operators

Dyadic operators are left-associative and are less binding than monadic operators.

- $e1 + e2$ Integer addition.
- $e1 - e2$ Integer subtraction.
- $e1 * e2$ Integer multiplication.
- $e1 \% e2$ Integer division.
- $e1 \& e2$ Bitwise conjunction.
- $e1 | e2$ Bitwise disjunction.
- $e1 \wedge e2$ Remainder after division of $e1$ by $e2$.
- $e1 \# e2$ $E1$ rounded up to the next multiple of $e2$.

Commands

Most commands consist of a verb followed by a modifier or list of modifiers. The following verbs are available. (The commands '?' and '/' may be followed by '*'; see *Addresses* for further details.)

- ?f Locations starting at text *address* in *objfil* are printed according to the format *f*.
- lf Locations starting at data *address* in *corefile* are printed according to the format *f*.
- =f The value of *address* itself is printed in the styles indicated by the format *f*. (For *i* format '?' is printed for the parts of the instruction that reference subsequent words.)

A *format* consists of one or more characters that specify a style of printing. Each format character may be preceded by a decimal integer that is a repeat count for the format character. While stepping through a format *dot* is incremented temporarily by the amount given for each format letter. If no format is given then the last format is used. The format letters available are as follows:

- o 2 Prints 2 bytes in octal. All octal numbers output by *adb* are preceded by 0.
- O 4 Prints 4 bytes in octal.
- q 2 Prints in signed octal.

Q 4	Prints long signed octal.
d 2	Prints in decimal.
D 4	Prints long decimal.
x 2	Prints 2 bytes in hexadecimal.
X 4	Prints 4 bytes in hexadecimal.
u 2	Prints as an unsigned decimal number.
U 4	Prints long unsigned decimal.
f 4	Prints the 32 bit value as a floating point number.
F 8	Prints double floating point.
b 1	Prints the addressed byte in octal.
c 1	Prints the addressed character.
C 1	Prints the addressed character using the following escape convention. Character values 000 to 040 are printed as an at-sign (@) followed by the corresponding character in the octal range 0100 to 0140. The at-sign character itself is printed as @@.
s <i>n</i>	Prints the addressed characters until a zero character is reached.
S <i>n</i>	Prints a string using the at-sign (@) escape convention. Here <i>n</i> is the length of the string including its zero terminator.
Y 4	Prints 4 bytes in date format (see <i>ctime</i> (S)).
i <i>n</i>	Prints as machine instructions. <i>n</i> is the number of bytes occupied by the instruction. This style of printing causes variables 1 and 2 to be set to the offset parts of the source and destination respectively.
a 0	Prints the value of <i>dot</i> in symbolic form. Symbols are checked to ensure that they have an appropriate type as indicated below. / local or global data symbol ? local or global text symbol = local or global absolute symbol
A 0	Prints the value of <i>dot</i> in absolute form.
p 2	Prints the addressed value in symbolic form using the same rules for symbol lookup as a .
t 0	When preceded by an integer, tabs to the next appropriate tab stop. For example, 8t moves to the next 8-space tab stop.
r 0	Prints a space.
n 0	Prints a newline.
"..." 0	Prints the enclosed string.

- ^ Decrements *dot* by the current increment. Nothing is printed.
- + Increments *dot* by 1. Nothing is printed.
- Decrements *dot* by 1. Nothing is printed.

newline

If the previous command temporarily incremented *dot*, makes the increment permanent. Repeat the previous command with a *count* of 1.

[?/] value mask

Words starting at *dot* are masked with *mask* and compared with *value* until a match is found. If **L** is used then the match is for 4 bytes at a time instead of 2. If no match is found then *dot* is unchanged; otherwise *dot* is set to the matched location. If *mask* is omitted then -1 is used.

[?/]w value ...

Writes the 2-byte *value* into the addressed location. If the command is **W**, writes 4 bytes. Odd addresses are not allowed when writing to the subprocess address space.

[?/]m segnum fpos size

Sets new values for the given segment's file position and size. If *size* is not given, then only the file position is changed. The *segnum* must be the segment number of a segment already in the memory map (see *Addresses*). If **?** is given, a text segment is affected; if **/** a data segment.

[?/]M segnum fpos size

Creates a new segment in the memory map. The segment is given file position *fpos* and physical size *size*. The *segnum* must not already exist in the memory map. If **?** is given, a text segment is created; if **/** a data segment.

>name

dot is assigned to the variable or register named.

! A shell is called to read the rest of the line following '!'.

\$modifier

Miscellaneous commands. The available *modifiers* are:

- <*f* Read commands from the file *f* and return.
- >*f* Send output to the file *f*, which is created if it does not exist.
- r** Print the general registers and the instruction addressed by **ip**.
Dot is set to **ip**.
- f** Print the floating registers in single or double length.

- b Print all breakpoints and their associated counts and commands.
- c C stack backtrace. If *address* is given then it is taken as the address of the current frame (instead of **bp**). If *count* is given then only the first *count* frames are printed. This command is case insensitive.
- e The names and values of external variables are printed.
- w Set the page width for output to *address* (default 80).
- s Set the limit for symbol matches to *address* (default 255).
- o Sets input and output default format to octal.
- d Sets input and output default format to decimal.
- x Sets input and output default format to hexadecimal.
- q Exit from *adb*.
- v Print all non zero variables in octal.
- m Print the address map.

:modifier

Manage a subprocess. Available modifiers are:

brc

Set breakpoint at *address*. The breakpoint is executed *count*-1 times before causing a stop. Each time the breakpoint is encountered the command *c* is executed. If this command sets *dot* to zero then the breakpoint causes a stop.

- dl Delete breakpoint at *address*.

r [*arguments*]

Run *obifil* as a subprocess. If *address* is given explicitly then the program is entered at this point; otherwise the program is entered at its standard entry point. *count* specifies how many breakpoints are to be ignored before stopping. *arguments* to the subprocess may be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command. All signals are turned on on entry to the subprocess.

R [*arguments*]

Same as the *r* command except that *arguments* are passed through a shell before being passed to the program. This means shell metacharacters can be used in filenames.

cos

The subprocess is continued and signal *s* is passed to it, see *signal(S)*. If *address* is given then the subprocess is continued at this address. If no signal is specified then the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for *r*.

ss As for **co** except that the subprocess is single stepped *count* times. If there is no current subprocess then *objfil* is run as a subprocess as for **r**. In this case no signal can be sent; the remainder of the line is treated as arguments to the subprocess.

k The current subprocess, if any, is terminated.

Variables

adb provides a number of variables. Named variables are set initially by *adb* but are not used subsequently. Numbered variables are reserved for communication as follows.

- 0 The last value printed.
- 1 The last offset part of an instruction source.
- 2 The previous value of variable 1.

On entry the following are set from the system header in the *corefile*. If *corefile* does not appear to be a **core** file then these values are set from *objfil*:

- b The base address of the data segment.
- d The data segment size.
- e The entry point.
- m The execution type.
- n The number of segments.
- s The stack segment size.
- t The text segment size.

Addresses

Addresses in *adb* refer to either a location in a file or in actual memory. When there is no current process in memory, *adb* addresses are computed as file locations, and requested text and data are read from the *objfil* and *corefile* files. When there is a process, such as after a **:r** command, addresses are computed as actual memory locations.

All text and data segments in a program have associated memory map entries. Each entry has a unique *segment number*. In addition, each entry has the *file position* of that segment's first byte, and the *physical size* of the segment in the file. When a process is running, a segment's entry has a *virtual size* which defines the size of the segment in memory at the current time. This size can change during execution.

When an address is given and no process is running, the file location corresponding to the address is calculated as:

$$\text{effective-file-address} = \text{file-position} + \text{offset}$$

If a process is running, the memory location is simply the offset in the given segment. These addresses are valid if and only if

$$0 \leq \text{offset} \leq \text{size}$$

where *size* is physical size for file locations and virtual size for memory locations. Otherwise, the requested *address* is not legal.

The initial setting of both mappings is suitable for normal **a.out** and **core** files. If either file is not of the kind expected then, for that file, *file position* is set to 0, and *size* is set to the maximum file size. In this way, the whole file can be examined with no address translation.

So that *adb* may be used on large files, all appropriate values are kept as signed 32 bit integers.

Files

a.out
core

See Also

ptrace(S), a.out(F), core(F)

Diagnostics

The message “adb” appears when there is no current command or format.

Comments about inaccessible files, syntax errors, abnormal termination of commands, etc.

Exit status is 0, unless last command failed or returned nonzero status.

Notes

A breakpoint set at the entry point is not effective on initial entry to the program.

System calls cannot be single stepped.

Local variables whose names are the same as an external variable may foul up the accessing of the external.

COFF or x.out format files are accepted and read transparently.

admin

create and administer SCCS files

Syntax

```
admin [-n] [-i[name]] [-rrel] [-t[name]] [-fflag[flag-val]]
[-dflag[flag-val]] [-alogin] [-eloglein] [-m[mrlist]] [-y[comment]] [-h]
[-z] files
```

Description

The *admin* command is used to create new SCCS files and change parameters of existing ones. Arguments to *admin*, which may appear in any order, consist of keyletter arguments that begin with a hyphen (-), and named files (note that SCCS file names must begin with the characters s.). If a named file does not exist, it is created, and its parameters are initialized according to the specified keyletter arguments. Parameters not initialized by a keyletter argument are assigned a default value. If a named file does exist, parameters corresponding to specified keyletter arguments are changed, and other parameters are left as is.

If a directory is named, *admin* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed since the effects of the arguments apply independently to each named file.

- n** This keyletter indicates that a new SCCS file is to be created.
- i[name]** The *name* of a file from which the text for a new SCCS file is to be taken. The text constitutes the first delta of the file (see **-r** keyletter for delta numbering scheme). If the **-i** keyletter is used but the file name is omitted, the text is obtained by reading the standard input until an end-of-file is encountered. If this keyletter is omitted, then the SCCS file is created empty. Only one SCCS file may be created by an *admin* command on which the **-i** keyletter is supplied. Using a single *admin* to create two or more SCCS files requires that they be created empty (no **-i** keyletter). Note that the **-i** keyletter implies the **-n** keyletter.

- rrel** The release into which the initial delta is inserted. This keyletter may be used only if the **-i** keyletter is also used. If the **-r** keyletter is not used, the initial delta is inserted into release 1. The level of the initial delta is always 1 (by default initial deltas are named 1.1).
- t[name]** The *name* of a file from which descriptive text for the SCCS file is to be taken. If the **-t** keyletter is used and *admin* is creating a new SCCS file (the **-n** and/or **-i** keyletters also used), the descriptive text file name must also be supplied. In the case of existing SCCS files: (C) a **-t** keyletter without a file name causes removal of descriptive text (if any) currently in the SCCS file, and (S) a **-t** keyletter with a file name causes text (if any) in the named file to replace the descriptive text (if any) currently in the SCCS file.
- fflag** This keyletter specifies a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file. Several **f** keyletters may be supplied on a single *admin* command line. The allowable *flags* and their values are:
- b** Allows use of the **-b** keyletter on a *get*(CP) command to create branch deltas.
 - cceil** The highest release (i.e., "ceiling"), a number greater than 0 but less than or equal to 9999, which may be retrieved by a *get*(CP) command for editing. The default value for an unspecified **c** flag is 9999.
 - ffloor** The lowest release (i.e., "floor"), a number greater than 0 but less than 9999, which may be retrieved by a *get*(CP) command for editing. The default value for an unspecified **f** flag is 1.
 - dSID** The default delta number (SIDs+1) to be used by a *get*(CP) command.
 - i[str]** Causes the "No id keywords (ge6)" message issued by *get*(CP) or *delta*(CP) to be treated as a fatal error. In the absence of this flag, the message is only a warning. The message is issued if no SCCS identification keywords (see *get*(CP)) are found in the text retrieved or stored in the SCCS file. If a value is supplied, the keywords must exactly match the given string; however, the string must contain a keyword and no embedded newlines.

- j** Allows concurrent *get*(CP) commands for editing on the same SID of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file.
- l***list* A *list* of releases to which deltas can no longer be made (*get -e* against one of these "locked" releases fails). The *list* has the following syntax:
- ```
<list> ::= <range> | <list> , <range>
<range> ::= | a
```
- The character **a** in the *list* is equivalent to specifying all releases for the named SCCS file.
- n** Causes *delta*(CP) to create a "null" delta in each of those releases (if any) being skipped when a delta is made in a new release (e.g., in making delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas serve as "anchor points" so that branch deltas may later be created from them. The absence of this flag causes skipped releases to be nonexistent in the SCCS file, preventing branch deltas from being created from them in the future.
- q***text* User-definable text substituted for all occurrences of the %Q% keyword in SCCS file text retrieved by *get*(CP).
- m***mod* Module name of the SCCS file substituted for all occurrences of the %M% keyword in SCCS file text retrieved by *get*(CP). If the **m** flag is not specified, the value assigned is the name of the SCCS file with the leading **s**. removed.
- t***type* Type of module in the SCCS file substituted for all occurrences of %Y% keyword in SCCS file text retrieved by *get*(CP).
- v***pgm* Causes *delta*(CP) to prompt for Modification Request (MR) numbers as the reason for creating a delta. The optional value specifies the name of an MR number validity checking program (see *delta*(CP)). (If this flag is set when creating an SCCS file, the **m** keyletter must also be used even if its value is null.)



- dflag Causes removal (deletion) of the specified *flag* from an SCCS file. The -d keyletter may be specified only when processing existing SCCS files. Several -d keyletters may be supplied on a single *admin* command. See the -f keyletter for allowable *flag* names.
  
- l*list* A *list* of releases to be "unlocked." See the -f keyletter for a description of the l flag and the syntax of a *list*.
  
- a*login* A *login* name or numerical system group ID to be added to the list of users which may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all *login* names common to that group ID. Several a keyletters may be used on a single *admin* command line. As many *logins* or numerical group IDs as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas. If *login* or group ID is preceded by a ! it is to be denied permission to make deltas.
  
- e*login* A *login* name or numerical group ID to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all *login* names common to that group ID. Several e keyletters may be used on a single *admin* command line.
  
- m[mr*list*] The list of Modification Requests (MR) numbers is inserted into the SCCS file as the reason for creating the initial delta in a manner identical to *delta*(CP). The v flag must be set; the MR numbers are validated if the v flag has a value (the name of an MR number validation program). Diagnostics will occur if the v flag is not set or MR validation fails.
  
- y[*comment*] The *comment* text is inserted into the SCCS file as a comment for the initial delta in a manner identical to that of *delta*(CP). Omission of the -y keyletter results in a default comment line being inserted in the form:  
     date and time created YY/MM/DD HH:MM:SS by *login*  
 The -y keyletter is valid only if the -i and/or -n keyletters are specified (that is, a new SCCS file is being created).
  
- h Causes *admin* to check the structure of the SCCS file and to compare a newly computed check-sum (the sum of all the characters in the SCCS file except those in the first line) with the check-sum that is stored in the first line of the SCCS file. Appropriate error diagnostics are produced.

This keyletter inhibits writing on the file, so that it nullifies the effect of any other keyletters supplied, and is, therefore, only meaningful when processing existing files.

**-z**

The SCCS file check-sum is recomputed and stored in the first line of the SCCS file (see **-h**, above).

Note that use of this keyletter on a truly corrupted file may prevent future detection of the corruption.

The last component of all SCCS file names must be of the form *s,filename*. New SCCS files are given mode 444 (see *chmod(S)*). Write permission in the pertinent directory is, of course, required to create a file. All writing done by *admin* is to a temporary x-file, called *x,filename* (see *get(CP)*) created with mode 444 if the *admin* command is creating a new SCCS file, or with the same mode as the SCCS file if it exists. After successful execution of *admin*, the SCCS file is removed (if it exists), and the x-file is renamed with the name of the SCCS file. This ensures that changes are made to the SCCS file only if no errors occurred.

It is recommended that directories containing SCCS files be mode 755 and that SCCS files themselves be mode 444. The mode of the directories allows only the owner to modify SCCS files contained in the directories. The mode of the SCCS files prevents any modification at all except by SCCS commands.

If it should be necessary to patch an SCCS file for any reason, the mode may be changed to 644 by the owner allowing use of *ed(C)*. *Care must be taken!* The edited file should *always* be processed by an **admin -h** to check for corruption followed by an **admin -z** to generate a proper check-sum. Another **admin -h** is recommended to ensure the SCCS file is valid.

The *admin* command also makes use of a transient lock file (called *z,filename*), which is used to prevent simultaneous updates to the SCCS file by different users. See *get(CP)* for further information.

## Files

---

|        |                                                                                             |
|--------|---------------------------------------------------------------------------------------------|
| g-file | Existed before the execution of <i>delta</i> ; removed after completion of <i>delta</i> .   |
| p-file | Existed before the execution of <i>delta</i> ; may exist after completion of <i>delta</i> . |
| q-file | Created during the execution of <i>delta</i> ; removed after completion of <i>delta</i> .   |



- x-file            Created during the execution of *delta*; renamed to SCCS file after completion of *delta*.
- z-file            Created during the execution of *delta*; removed during the execution of *delta*.
- d-file            Created during the execution of *delta*; removed after completion of *delta*.
- /usr/bin/bdiff   Program to compute differences between the "gotten" file and the *g-file*.

## See Also

---

delta(CP), get(CP), prs(CP), what(CP), sccsfile(F), ed(C)

## Standards Conformance

---

*admin* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## ar

---

archive and library maintainer for portable archives

### Syntax

---

**ar** *key* [*keyarg*] [*posname*] *afile* [*name*] ...

### Description

---

The *ar* command maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the link editor. It can be used, though, for any similar purpose. The magic string and the file headers used by *ar* consist of printable ASCII characters. If an archive is composed of printable files, the entire archive is printable. Archives of text files created by *ar* are portable between implementations of System V.

When *ar* creates an archive, it creates headers in a format that is portable across all machines. The portable archive format and structure is described in detail in *ar(F)*. The archive symbol table (described in *ar(F)*) is used by the link editor (*ld(CP)*) to effect multiple passes over libraries of object files in an efficient manner. An archive symbol table is only created and maintained by *ar* when there is at least one object file in the archive. The archive symbol table is in a specially named file that is always the first file in the archive. This file is never mentioned nor is it accessible to the user. Whenever the *ar(CP)* command is used to create or update the contents of such an archive, the symbol table is rebuilt. The *s* option, described in the following text, will force the symbol table to be rebuilt.

Unlike command options, the command *key* is a required part of *ar*'s command line. The *key* (which may begin with a -) is formed with one of the following letters: **drqtpmx**. Arguments to the *key*, alternatively, are made with one or more of the following set: **vuaibcls**. *posname* is an archive member name used as a reference point in positioning other files in the archive. *afile* is the archive file. The *names* are constituent files in the archive file. The meanings of the *key* characters are as follows:

- d** Delete the named files from the archive file.
- r** Replace the named files in the archive file. If the optional character **u** is used with **r**, then only those files with dates of modification later than the archive files are replaced. If an optional positioning character from the set **abi** is used, then the *posname* argument must be present and specify that new files are to be placed after (**a**) or before (**b** or **i**) *posname*. Otherwise new files are placed at the end.



- q** Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive. This option is useful to avoid quadratic behavior when creating a large archive piece-by-piece. Unchecked, the file may grow exponentially up to the second degree.
- t** Print a table of contents of the archive file. If no names are given, all files in the archive are tabled. If names are given, only those files are tabled.
- p** Print the named files in the archive.
- m** Move the named files to the end of the archive. If a positioning character is present, then the *posname* argument must be present and, as in **r**, specify where the files are to be moved.
- x** Extract the named files. If no names are given, all files in the archive are extracted. In neither case does **x** alter the archive file.

The meanings of the key arguments are as follows:

- v** Give a verbose file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t**, give a long listing of all information about the files. When used with **x**, precede each file with a name.
- c** Suppress the message that is produced by default when *afile* is created.
- l** Place temporary files in the local (current working) directory rather than in the default temporary directory, *TMPDIR*.
- s** Force the regeneration of the archive symbol table even if *ar*(CP) is not invoked with a command which will modify the archive contents. This command is useful to restore the archive symbol table after the *strip*(CP) command has been used on the archive.

## Files

---

*\$TMPDIR*/\*                      temporary files

*\$TMPDIR* is usually */usr/tmp* but can be redefined by setting the environment variable **TMPDIR** [see *tempnam()* in *tempnam(S)*].

## See Also

---

*ld*(CP), *lorder*(CP), *strip*(CP), *tsort*(CP), *tmpnam*(S), *a.out*(F), *ar*(F).

## Notes

---

If you are using XENIX binaries, please refer to the manual entry for this utility in the *XENIX Development Guide* for information on the appropriate usage with XENIX binaries.

If the same file is mentioned twice in an argument list, it may be put in the archive twice.

## Standards Conformance

---

*ar* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



# as

---

common assembler

## Syntax

---

as [options] filename

## Description

---

The *as* command assembles the named file. The following flags may be specified in any order:

- o *objfile* Put the output of the assembly in *objfile*. By default, the output file name is formed by removing the *.s* suffix, if there is one, from the input file name and appending a *.o* suffix.
- n Turn off long/short address optimization. By default, address optimization takes place.
- m Run the *m4* macro processor on the input to the assembler.
- R Remove (unlink) the input file after assembly is completed.
- dl Do not produce line number information in the object file.
- V Write the version number of the assembler being run on the standard error output.
- Y [*md*],*dir* Find the **m4** preprocessor (**m**) and/or the file of predefined macros (**d**) in directory *dir* instead of in the customary place.

## Files

---

*TMPDIR*/\* temporary files

*TMPDIR* is usually */usr/tmp* but can be redefined by setting the environment variable *TMPDIR* [see *tempnam()* in *tempnam(S)*].

## See Also

---

cc(CP), ld(CP), m4(CP), nm(CP), strip(CP), tmpnam(S), a.out(F).

## Warning

---

If the **-m** (*m4* macro processor invocation) option is used, keywords for *m4* (see *m4*(CP)) cannot be used as symbols (variables, functions, labels) in the input file since *m4* cannot determine which are assembler symbols and which are real *m4* macros.

## Notes

---

The **.align** assembler directive may not work in the **.text** section when optimization is performed.

Arithmetic expressions may only have one forward referenced symbol per expression.

Wherever possible, the assembler should be accessed through a compilation system interface program (such as *cc*(CP)).

## Standards Conformance

---

*as* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



## cb

---

C program beautifier

### Syntax

---

`cb [-s] [-j] [-l leng] [file ...]`

### Description

---

The *cb* command reads C programs either from its arguments or from the standard input and writes them on the standard output with spacing and indentation that display the structure of the code. Under default options, *cb* preserves all user new-lines.

The *cb* command accepts the following options:

- s            Canonicalizes the code to the style of Kernighan and Ritchie in "*The C Programming Language*".
- j            Causes split lines to be put back together.
- l *leng*      Causes *cb* to split lines that are longer than *leng*.

### See Also

---

cc(CP)

Kernighan, B. W., and Ritchie, D. M., *The C Programming Language*, Prentice-Hall, 1978.

### Notes

---

Punctuation that is hidden in preprocessor statements will cause indentation errors.

## CC

---

invokes the C compiler.

## Syntax

---

`cc [option]...file...[option...file...] [-link[link-libinfo]]`

## Description

---

`cc` is the C compiler command. It creates executable programs by compiling and linking the files named by the *filename* arguments. `cc` copies the resulting program to the file **a.out**. The file **a.out** is in COFF format by default. To specify the generation of standard XENIX files, (OMF format files), use the **-xenix**, **-x2.3**, or **-xout** flags. See the section on creating XENIX binaries in the *Xenix Development and Portability Guide* for more information.

Additionally, you can direct the compiler to produce DOS or OS/2 format binaries by giving the **-dos** or **-os2** flags. Note that if these flags are present, they override all other flags that may be given on the command line. Thus, if you place both the **-xenix** and the **-dos** flags on your command line, the output will be in DOS format. For more information on developing DOS and OS/2 binaries, see the *DOS-OS/2 Development Guide*.

The *filename* can name any C or assembly language source file or any object or library file. C source files must have a **.c** filename extension. Assembly language source files must have **.s** or **.asm**, object files **.o**, and library files **.a** extensions. `cc` invokes the C compiler for each C source file and copies the result to an object file whose basename is the same as the source file but whose extension is **.o**. The `cc` command invokes the AT&T assembler, *as*, for each assembly source file that ends with **.s**, and it invokes the Microsoft assembler, *masm*, for each assembly source file that ends with **.asm**. The `cc` command copies the result to an object file with extension **.o**. `cc` ignores object and library files until all source files have been compiled or assembled. It then invokes the link editor, *ld*, and combines all the object files it has created together with object files and libraries given in the command line to form a single program.

Files are compiled as they occur on the command line, and only flags that are listed before the name of the file on the command line are taken into account. This makes the order of files very important. The exception to this rule are flags to the linker, such as **-l**, which are taken into account during linking after all objects to be created are gathered. Library files are examined only if functions referenced in previous files have not yet been defined. Only those functions that define unresolved references are concatenated. A number of "standard"



libraries are searched automatically. These libraries support the standard C library functions and program startup routines. Which libraries are used depends on the program's specific flags and memory model (see the options and "Memory Models" below). By default, the libraries are found in the directory */lib* and their names are prefixed with "lib". These libraries are generally used for resolving references during linking. The entry point of the resulting program is set to the beginning of the standard startup code which then calls the "main()" function of the program.

By default, **cc** creates binaries with full International functionality; the library file, **libintl**, is automatically included in the link command line, and **M\_INTERNAT** is automatically defined by **cc**. The **-nointl** option cancels international functionality. For more information, refer to the section on *Developing International Applications*.

There are the following options:

**-c** Creates a linkable x.out object file for each source file but does not link these files. No executable program is created.

**-compat**

Makes an executable file that is binary compatible across the following systems (as distributed by certain vendors):

386 UNIX System V Release 3.2

-286 System V

-386 System V

-286 3.0

-8086 System V

The XENIX libraries are used with this option.

**-C** Preserves comments when preprocessing a file with **-E**, **-P**, or **-EP**. That is, comments are not removed from the preprocessed source. This option may only be used in conjunction with **-E**, **-P**, or **-EP**.

**-CSON, -CSOFF**

When optimization (**-O**) is also specified, these options enable or disable "common sub-expression" optimization.

**-d** Displays the various passes and their arguments before they are executed.

**-dos**

Directs **cc** to create an executable program for MS-DOS systems. The DOS libraries are used with this option.

**-Dname[=string]**

Defines *name* to the preprocessor as if defined by #define in each source file. The form “-Dname” sets *name* to a value of 1. The form “-Dname=string” sets *name* to the given *string*. See the -U option for more information on the default manifest defines.

**-E** Preprocesses each source file as described for **-P**, but copies the result to the standard output. The option also places a #line directive with the current input line number and source file name at the beginning of output for each file.**-EP**

Preprocesses each source file as described for **-P**, and prints the output both to a file and the standard output -**EP** does not place a #line directive at the beginning of the file.

**-Fnum**

Sets the size of the program stack to *num* bytes. The value of *num* must be given in hexadecimal. The default stack for the 8086 is variable, starting at the top of a full 64 Kbyte data segment that grows down until it reaches data. The default stack for the 80286 is 1000 bytes (0xFFFF hexadecimal). This option does not apply to the 80386, which has a variable stack.

**-Fa, -Faname**

Create an assembly source listing in source.asm or the named file. Continues with the link if requested. This option performs the same function as the -S option.

**-Fc, -Fcname**

Create a merged assembler and C listing in source.L or in the named file.

**-Fename**

Names the executable program file *name*.

**-Fl, -Flname**

Create a listing file in source.L (or the named file) with assembly source and object code. Continues with the link if requested.

**-Fm, -Fmname**

Instruct the linker to create a map listing in a file called a.map (or the named file). This file contains the names of all segments in order of their appearance in the load module. Use this option only to produce XENIX compatible binaries.

**-Foname**

The object filename will be *name* instead of source.o.



**-FPa, -FPc, -FPc87, -FPi, -FPi87**

When used in conjunction with **-dos** or **-os2**, these options control the type of floating point code generated and which library support to use. The default is **-FPi**. For more information see the *DOS/OS/2 Cross Development Guide*.

**-Fs, -Fsname**

Creates a C source listing in *source.S* or the named file.

**-g** Includes information for the symbolic debugger (sdb) and for CodeView. (This is equivalent to the **-Zi** option.)**-Gkeyword**

Specifies the alternative calling sequence and naming conventions used in System V 386 Pascal and System V 386 FORTRAN. There are three keywords that go with this option. They are **fortran**, **pascal**, and **cdecl**. For more information about this option, see the *C User's Guide*.

**-Gs**

Removes stack probe routines. This option reduces binary size and speeds execution slightly. Use of this option disables stack checking and stack overflow can occur without warning or error messages.

**-Hnum**

Sets the maximum length of external symbols to *num*. This option is equivalent to the **-nl** option.

**-help**

Prints help menu.

**-HELP**

Same as **-help**.

**-i** Creates separate instruction and data spaces for small model programs. When the output file is executed, the program text and data areas are allocated separate physical segments. The text portion will be read-only and may be shared by all users executing the file. This option is implied when creating middle or large model programs.**-I pathname**

Adds *pathname* to the list of directories to be searched when an **#include** file is not found in the directory containing the current source file or whenever angle brackets (<>) enclose the filename. If the file cannot be found in directories in this list, directories in a standard list are searched.

**-J** Changes the default mode for the *char* type to unsigned.

**-K** Removes stack probes from a program. Stack probes are used to detect stack overflow on entry to program routines. Code generated for the 80386 processor does not require stack probes, therefore this option has no effect if **-M3** is specified.

**-lname**

Searches library *name* for unresolved function references.

**-link**

Lets the user specify linker switches at compile time that are not directly supported by the compiler. This option must be specified last on the **cc** command line. All text (options and filenames) that follows this option is passed directly to the **ld** linker. (Note that all options not recognized by the compiler are passed to the linker.)

**-L** Creates an assembler listing file containing assembled code and assembly source instructions. The listing is made in a file whose basename is the same as the source but whose extension is **.L**. This option suppresses the **-S** option.

**-LARGE**

Invokes the large model passes of the compiler. This option is included in this release only for backward compatibility.

**-m name**

Creates a map file called *name*. This option is equivalent to the **-Fm** option. Use this option only to produce XENIX compatible binaries.

**-M string**

Sets the program configuration. This configuration defines the program's memory model, word order, and data threshold. It also enables C language enhancements such as advanced instruction set and keywords. Note that the number specifying the CPU type (0, 1, 2, or 3) must be given before the letter specifying the size model of the program. For example, to compile the program to be large model 286, the flag should be constructed as follows:

**-M2l**

The *string* may be any combination of the following ("s", "m", "l", and "h" are mutually exclusive):

- |   |                                                                           |
|---|---------------------------------------------------------------------------|
| a | Restricts the language to ansi specifications.                            |
| c | Creates a compact model program.<br><b>For 286 compilations only.</b>     |
| s | Creates a small model program (default).                                  |
| m | Creates a middle model program.<br><b>For 8086/286 compilations only.</b> |



- l Creates a large model program.  
**For 8086/286 compilations only.**
- h Creates a huge model program.  
**For 286 compilations only.**
- e Enables the far, near, huge, pascal, and fortran keywords. Also enables certain non-ANSI extensions necessary to ensure compatibility with existing versions of the C compiler (this applies only to versions of the C compiler that support ANSI C).
- 0 Enables 8086 code generation. The "s", "m", and "l" specifiers are valid here.
- 1 Enables 186 code generation. The "s", "m", "l", and "h" specifiers are valid here.
- 2 Enables 286 code generation for compiled C source files. The "s", "m", "l", and "h" specifiers are valid here.
- 3 Enables 386 code generation. The "m", "l", and "h" specifiers are invalid here.
- b Reverses the word order for **long** types. High order word is first. Default is low order word first.
- t num Sets the threshold for the size of the largest data item in the data group to *num*. Default is 32,767. This option can only be used in large model programs.
- d Instructs the compiler not to assume register SS=DS.
- f Enables software floating point that does not exist in UNIX. Useful when compiling object files to be linked on MS-DOS.

Note that specifying CPU type with 0, 1, or 2 produces XENIX files and uses the XENIX libraries appropriate to the model size. See the **-xenix** option for more information.

**-n** Sets pure text model. Pure text without separate instruction and data space is only supported for 80386 binaries. For 8086, 80186, or 80286 binaries this option is equivalent to the **-i** option and will give a warning that it is setting **-i** when used.

**-nl num**

Sets the maximum length of external symbols to *num*. Names longer than *num* are truncated before being copied to the external symbol table.

**-nointl**

Directs **cc** to create a binary that does not include the international functionality.

**-ND name**

Sets the data segment name for each compiled or assembled source file to *name*. If **-ND** is not given, the name “\_DATA” is used.

In large model programs (**-MI**) the **-ND** option can only be used on “leaf modules” - those that make no calls to routines in another segment.

**-NM name**

Sets the module name for each compiled or assembled source file to *name*. If not given, the filename of each source file is used.

**-NT name**

Sets the text segment name for each compiled or assembled source file to *name*. If not given, the name “*module\_TEXT*” is used for middle model and “\_TEXT” for small model programs. This option should not be used on 386 code.

**-o filename**

Defines *filename* to be the name of the final executable program. This option overrides the default name **a.out**. *Filename* can not end in **.o** or **.c**.

**-os2**

Directs **cc** to create an executable program for OS/2. OS/2 libraries are used with this option.

**-O string**

Invokes the object code optimizer. The *string* consists of one or more of the following characters:

- d** Disables most optimization.
- a** Relaxes alias checking.
- s** Optimizes code for space.
- t** Optimizes code for speed.
- i** Expands certain intrinsic functions inline.
- x** Performs maximum optimization. Equivalent to **-Oactli**.
- c** Eliminates common expressions. (386 only)
- l** Performs various loop optimizations. (386 only)
- p** Precision optimization.

- p** Adds code for program profiling. Profiling code counts the number of calls to each routine in the program and copies this information to the **mon.out** file. This file can be examined using the *prof*(CP) command.

**-pack**

Packs structures. Each structure member is stored at the first available byte, without regard to *int* boundaries. Although this will save space, execution will be slower because of the extra time required to access 16 bit members that begin on odd boundaries.



- P Preprocesses each source file and copies the result to a file whose basename is the same as the source but whose extension is **.i**.
- r Performs an incremental linking action.
- s Instructs the linker to strip the symbol table information from the executable output file.
- S  
Creates an assembly source listing in a file whose basename is the same as the source but whose extension is **.asm**. It should be noted that this file is in MASM format. This option performs the same function as the **-Fa** option.
- Ss -St  
Sets subtitle (-Ss) and title (-St) for source listings and bypasses the linking operations of **cc**.
- SEG *num*  
Sets the maximum number of segments that the linker can handle to *num*, which can range from 1 to 1024. If 1024 is too small, use the **-NT** option to reduce the number of different segment names.
- Tc  
Tells **cc** that the given file is a C source file. Use this option if the C source file does not have the **.c** filename extension.
- u Eliminates all manifest defines. Also see **-U**.
- U *definition*  
Removes or undefines the given manifest define. The manifest defines are as follows:

```

M_I86
M_XENIX
M_UNIX
M_SYS3 or M_SYSIII
M_SYS5 or M_SYSV
M_BITFIELDS
M_WORDSWAP
M_SDATA or M_LDData
M_STEXT or M_LTEXT
M_I8086 or M_I186 or M_I286 or M_I386
M_I86SM or M_I86MM or M_I86LM
M_INTERNAT
i386
unix

```

**-Vstring**

This option allows you to place a text string in your object file. Typically, a copyright notice or the version of the program is specified here. *string* must be enclosed within double quotation marks if it contains blank spaces or quotation marks. A backslash character must also precede any quotation marks within the *string*.

**-w** Prevents compiler warning messages from being issued. Same as **“-W 0”**.

**-W num**

Sets the output level for compiler warning messages. If *num* is 0, no warning messages are issued. If 1, only warnings about program structure and overt type mismatches are issued. If 2, warnings about strong typing mismatches are issued. If 3, warnings for all automatic conversions are issued. This option does not affect compiler error message output.

**-xenix**

For XENIX Cross Development, this option produces XENIX programs (in OMF format) using the XENIX libraries and include files. These resulting programs are compatible with XENIX 386 System V OS. If used in conjunction with the **-M2** option, it is compatible with both 286 and 386 System V.

User libraries that are used for XENIX Cross Development must be in *ranlib*(CP) format; that is, the first member must be named *\_\_SYMDEF*, which is a dictionary for the library.

**-x2.3**

This option is equivalent to the **-xenix** option, but it additionally includes the extended functions available with XENIX System V 386 release 2.3. When used in conjunction with **-M2**, this option produces XENIX System V 286 2.3 compatible files.

**-X** Removes the standard directories from the list of directories to be searched for **#include** files.

**-z** Displays the various passes and their arguments but does not execute them.

**-Za**

Restricts the language to ansi specifications. This option is equivalent to the **-Ma** option.

**-Zd**

Includes line number information in the object file.



**-Ze**

Enables the far, near, huge, pascal, and fortran keywords. This option is equivalent to the **-Me** option.

**-Zg**

Generates function declarations from function definitions and writes declarations to standard output—omits p2, p3, and ld.

**-Zi**

Includes information used by the symbolic debugger (sdb) and by CodeView in the output file. (This is equivalent to the **-g** option.)

**-Zl**

Removes default library information from the object file.

**-Zpn**

Packs structure members in memory. Allocates alignment to 1 (for 8086 processors). The *n* argument can be 1, 2, or 4, where

- 1    allocates alignment to 1.
- 2    allocates alignment to 2 (default for 80286 programs).
- 4    allocates alignment to 4 (default for 80386 programs).

**-Zs**

Performs syntax check only—omits calling p2, p3, and ld.

Many options (or equivalent forms of these options) are passed to the link editor as the last phase of compilation. The **-M** option with the "s", "m", and "l" configuration options are passed to specify memory requirements. The **-i**, **-F**, and **-p** are passed to specify other characteristics of the final program.

The **-D** and **-I** options may be used several times on the command line. The **-D** option must not define the same name twice. These options affect subsequent source files only.

## Memory Models

---

---

### *Note*

These memory models are used for XENIX Cross Development only.

---

*cc* can create programs for five different memory models: small, middle, compact, large, and huge. In addition, small model programs can be pure or impure. On the 8086 and 80286 processors, these various segmentation models allow programs with code or data larger than 64K bytes. Since the 80386 can address segments larger than 64K bytes, the middle, large and huge models are not supported on the 80386.

#### Impure-Text Small Model

These programs occupy one 64K byte physical segment in which both text and data are combined. *cc* creates impure small model programs by default. They can also be created using the **-Ms** option.

#### Pure-Text Small Model

These programs occupy two 64K byte physical segments. Text and data are in separate segments. The text is read-only and may be shared by several processes at once. The maximum program size is 128 Kbytes. Pure small model programs are created using the **-i** and **-Ms** options.

#### Middle Model

These programs occupy several physical segments, but only one segment contains data. Text is divided among as many segments as required. Special calls and returns are used to access functions in other segments. Text can be any size. Data must not exceed 64K bytes. Middle models programs are created using the **-Mm** option. These programs are always pure.

#### Compact Model

This model of program has a maximum of 64K of text, but multiple segments of data. Data can be any size, but text must not exceed 64K. Compact model programs are created with the **-Mc** option.

#### Large Model

These programs occupy several physical segments with both text and data in as many segments as required. Special calls and returns are used to access functions in other segments. Special addresses are used to access data in other segments. Text and data may be any size, but no data item may be larger than 64K bytes. Large model programs are created using the **-Ml** option. These programs are always pure.

#### Huge Model

These programs occupy several physical segments with both text and data in as many segments as required. It is possible to allow a data construct that spans 64K byte segments. This implementation imposes limits on the way the data construct is put together and where it is located in memory. Huge model programs are created using the **-Mh** option. These programs are always pure.



Small, middle, large and huge model object files can only be linked with object and library files of the same model. It is not possible to combine small, medium, large, and huge model object files in one executable program. *cc* automatically selects the correct small, middle, large, or huge versions of the standard libraries based on the configuration option. It is up to users to make sure that all of their own object files and private libraries are properly compiled in the appropriate model.

The special calls and returns used in middle, large, and huge model programs may affect execution time. In particular, the execution time of a program which makes heavy use of functions and function pointers may differ noticeably from small model programs.

In middle, large, and huge model programs, function pointers are 32 bits long. In large and huge model programs, data pointers are 32 bits long. Programs making use of such pointers must be written carefully to avoid incorrect declaration and use of these variables.

The **-NM**, **-NT**, and **-ND** options may be used with middle, large, and huge model programs to direct the text and data of specific object files to named physical segments. All text having the same text segment name is placed in a single physical segment. Similarly, all data having the same data segment name is placed in a single physical segment.

*cc* reads **/etc/default/cc** to obtain information about default options and libraries. The default file may contain lines beginning with the following patterns:

FLAGS=

and

LIBS=

Any parameters following the **FLAGS=** pattern are treated by *cc* as if they had been specified at the start of the *cc* command line. Parameters following the **LIBS=** pattern are treated as if they had been specified at the end of the command line. This option is intended for, but not restricted to, the specification of additional libraries. *cc* always searches for a file in **/etc/default** that matches the last component of the pathname by which *cc* was invoked. Thus by linking *cc* to several different names and invoking it by those names, different defaults can be selected.

An example `/etc/default/cc` file follows:

FLAGS= -M2

LIBS= -lx

This invokes the large model versions of the compiler passes to generate 286 code with **far** and **near** keywords enabled, and includes **libx.a** on all links.

## Files

---

|                                            |                               |
|--------------------------------------------|-------------------------------|
| <code>/bin/cc</code>                       | Driver                        |
| <code>/lib/p1, p2, p3</code>               | 286 compiler passes           |
| <code>/lib/386/c_1p, p2_386, p3_386</code> | 386 compiler passes           |
| <code>/lib/*.a</code>                      | Standard libraries            |
| <code>/etc/default/cc</code>               | Default options and libraries |

## See Also

---

`ar(CP)`, `ld(CP)`, `lint(CP)`, `machine(HW)`, `masm(CP)`, `ranlib(CP)`,  
*C User's Guide*, *C Library Guide*, and *C Language Reference*

## Notes

---

Error messages are produced by the program that detects the error. These messages are usually produced by the C compiler, but may occasionally be produced by the assembler or the link loader.

All object module libraries must have a current *ranlib* directory. The user must make sure that the most recent library versions have been processed with *ranlib*(CP) before linking. If this is not done, *ld* cannot create executable programs using these libraries.

## Standards Conformance

---

`cc` is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



## **cdc**

change the delta commentary of an SCCS delta

### **Syntax**

**cdc -rSID [-m[mrlist]] [-y[comment]] files**

### **Description**

The *cdc* command changes the *delta commentary*, for the SID (SCCS identification string) specified by the **-r** keyletter, of each named SCCS file.

*Delta commentary* is defined to be the Modification Request (MR) and comment information normally specified via the *delta*(CP) command (**-m** and **-y** keyletters).

If a directory is named, *cdc* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of - is given, the standard input is read (see *Warnings*, below) and each line of the standard input is taken to be the name of an SCCS file to be processed.

Arguments to *cdc*, which may appear in any order, consist of *keyletter* arguments and file names.

All the described *keyletter* arguments apply independently to each named file:

**-rSID**           Used to specify the SCCS identification (SID) string of a delta for which the delta commentary is to be changed.

**-mmrlist**       If the SCCS file has the **v** flag set (see *admin*(CP)) then a list of MR numbers to be added and/or deleted in the delta commentary of the SID specified by the **-r** keyletter may be supplied. A null MR list has no effect.

MR entries are added to the list of MRs in the same manner as that of *delta*(CP). In order to delete an MR, precede the MR number with the character ! (see *Examples*, below). If the MR to be deleted is currently in the list of MRs, it is removed and changed into a "comment" line. A list of all deleted MRs is placed in the comment section of the delta commentary and preceded by a comment line stating that they were deleted.

If **-m** is not used and the standard input is a terminal, the prompt **MRs?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The **MRs?** prompt always precedes the **comments?** prompt (see **-y** keyletter).

**MRs** in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the **MR** list.

Note that if the **v** flag has a value (see *admin*(CP)), it is taken to be the name of a program (or shell procedure) which validates the correctness of the **MR** numbers. If a non-zero exit status is returned from the **MR** number validation program, *cdc* terminates and the delta commentary remains unchanged.

**-y[comment]** Arbitrary text used to replace the *comment*(s) already existing for the delta specified by the **-r** keyletter. The previous comments are kept and preceded by a comment line stating that they were changed. A null *comment* has no effect.

If **-y** is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the *comment* text.

Simply stated, the rules are:

(C) If you made the delta, you can change its delta commentary.

or

(S) If you own the file and directory, you can modify the delta commentary.



## Examples

---

```
cdc -r1.6 -m"bl78-12345 !bl77-54321 bl79-00001" -ytrouble
s.file
```

adds bl78-12345 and bl79-00001 to the MR list, removes bl77-54321 from the MR list, and adds the comment **trouble** to delta 1.6 of s.file.

```
cdc -r1.6 s.file
MRs? !bl77-54321 bl78-12345 bl79-00001
comments? trouble
```

does the same thing.

## Files

---

|        |                         |
|--------|-------------------------|
| x-file | (see <i>delta</i> (CP)) |
| z-file | (see <i>delta</i> (CP)) |

## See Also

---

admin(CP), delta(CP), get(CP), prs(CP), sccsfile(F).

## Warnings

---

If SCCS file names are supplied to the *cdc* command via the standard input (- on the command line), then the **-m** and **-y** keyletters must also be used.

## cfLOW

---

generate C flowgraph

### Syntax

---

**cfLOW** [-r] [-ix] [-i\_ ] [-dnum] files

### Description

---

The *cfLOW* command analyzes a collection of C, yacc, lex, assembler, and object files and attempts to build a graph charting the external references. Files suffixed with *.y*, *.l*, and *.c* are yacc'd, lex'd, and C-preprocessed as appropriate. The results of the preprocessed files, and files suffixed with *.i*, are then run through the first pass of *lint*(CP). Files suffixed with *.s* are assembled. Assembled files, and files suffixed with *.o*, have information extracted from their symbol tables. The results are collected and turned into a graph of external references which is displayed upon the standard output.

Each line of output begins with a reference number, followed by a suitable number of tabs indicating the level, then the name of the global symbol followed by a colon and its definition. Normally only function names that do not begin with an underscore are listed (see the *-i* options below). For information extracted from C source, the definition consists of an abstract type declaration (for example, **char \***), and, delimited by angle brackets, the name of the source file and the line number where the definition was found. Definitions extracted from object files indicate the file name and location counter under which the symbol appeared (for example, *text*). Leading underscores in C-style external names are deleted.

Once a definition of a name has been printed, subsequent references to that name contain only the reference number of the line where the definition may be found. For undefined references, only *<>* is printed.

As an example, given the following in *file.c*:



```

int i;

main()
{
 f();
 g();
 f();
}

f()
{
 i = h();
}

```

the command

```
cfow -ix file.c
```

produces the output

```

1 main: int(), <file.c 4>
2 f: int(), <file.c 11>
3 h: <>
4 i: int, <file.c 1>
5 g: <>

```

When the nesting level becomes too deep, the output of *cfow* can be piped to *pr(C)*, using the **-e** option, to compress the tab expansion to something less than every eight spaces.

In addition to the **-D**, **-I**, and **-U** options (which are interpreted just as they are by *cc(CP)* and *cpp(CP)*), the following options are interpreted by *cfow*:

- r** Reverse the “caller: callee” relationship producing an inverted listing showing the callers of each function. The listing is also sorted in lexicographical order by callee.
- ix** Include external and static data symbols. The default is to include only functions in the flowgraph.
- i\_** Include names that begin with an underscore. The default is to exclude these functions (and data if **-ix** is used).
- dnum** The *num* decimal integer indicates the depth at which the flowgraph is cut off. By default this is a very large number. Attempts to set the cutoff depth to a nonpositive integer will be ignored.

## See Also

as(CP), cc(CP), cpp(CP), lex(CP), lint(CP), nm(CP), yacc(CP), pr(C)

## Diagnostics

---

*cflow* complains about bad options. It also complains about multiple definitions and only believes the first one. Other messages may come from the various programs used (for example, the C-preprocessor).

## Notes

---

Files produced by *lex*(CP) and *yacc*(CP) cause the reordering of line number declarations which can confuse *cflow*. To get proper results, feed *cflow* the *yacc* or *lex* input.

## Standards Conformance

---

*cflow* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



# chkshlib

## compare shared libraries tool

### Syntax

**chkshlib** [-b] [-i] [-n] [-v] file1 [file2 file3 ... ]

### Description

*chkshlib* checks for compatibility between files. Input files can be combinations of host shared libraries, non-stripped target shared libraries, and non-stripped executable files. A file is compatible with another file if every library symbol in it that should be matched is matched in the second (that is, the symbol exists and has the same address in both files). The path name for the target shared library in both files must be identical (unless the **-i** option is set).

It is possible for *file1* to be compatible with *file2* without the reverse also being true.

If one incompatibility is found it is reported to stdout and processing stops (unless the **-v** option is set).

The options to *chkshlib* are:

- v** Cause verbose reporting of all incompatibilities to stdout.
- b** If there are symbols found in *file1* that are not in the bounds of *file2*, report warning messages to stderr.
- i** Turn off the restriction that the path names for the target shared library need to be identical for two files to be compatible.
- n** Indicate that there are exactly two input files, which are target shared libraries, where the first references symbols in the second ("includes" the second).

The output of *chkshlib* depends upon the input. If the first input file is an executable file and the other input files, if any, are target shared libraries, the output states whether or not the executable file can execute using each target shared library. If there are no target shared libraries supplied, *chkshlib* performs the compatibility check against the target shared libraries specified in the **.lib** section of the executable file.

If the first input file is an executable file and the other input file(s) is a host shared library, the output states whether or not the executable file could have been produced using each host.

If one input file is a host shared library and the other input file, if any, is a target shared library, the output states whether or not the host shared library could produce executable files that will run with the target shared library. If no target shared library is supplied, then *chkshlib* performs the compatibility check against the target specified in the *.lib* section of the library definition file found in the host.

If both input files are target shared libraries or both input files are host shared libraries, the output states whether or not the first file could replace the second and vice versa.

If both input files are target libraries and the **-n** option is set, the output states if the first file references symbols in the second file ("includes" the second).

Compatibility of all other combinations of host shared libraries, target shared libraries, and executable files has no useful meaning, and these other combinations of files are not accepted as valid input to *chkshlib*.

## See Also

---

mkshlib(CP)

## Diagnostics

---

Exit status is 0 if no incompatibilities are found, 1 if an incompatibility is found, and 2 if a processing error occurs.

## Notes

---

*chkshlib* requires that you use the **-i** option whenever you use the **-n** option.

Standard binaries distributed with the system are stripped, and *chkshlib* cannot be used with them.



# codeview

---

## visual debugger

### Syntax

---

`cv` [*options*] *executable\_file* [*arguments*]

### Description

---

The *codeview* debugger is a screen-oriented debugger. CodeView displays source code or assembly code, indicates which line is about to be executed, dynamically watches the values of variables (local or global), switches screens to display program output, and performs many other related functions. You can use CodeView to debug programs written in FORTRAN, BASIC, or Pascal as well as assembly language. CodeView displays in either window mode or in sequential mode. In window mode, pull-down menus and function keys offer fast access to the most common commands.

The *executable file* is created from compiled object files. It must be compiled and linked with the correct options so that it contains the line-number information and a symbol table needed by CodeView. The correct options for compiling and linking for use with CodeView are described in *The CodeView Debugger*.

The optional *arguments* are parameters passed to the *executable file*. If the program you are debugging does not accept command-line arguments, you do not need to pass any arguments.

The following options are available:

**-c*commands***

Executes one or more *commands* automatically upon start-up.

- t** Starts the debugger in sequential mode, rather than in the default window mode.

For information about using the CodeView debugger, see *The CodeView Debugger*.

### See Also

---

adb(CP), sdb(CP)

**Value Added**

---

*CodeView* is an extension of AT&T System V provided by the Santa Cruz Operation.



## comb

---

combine SCCS deltas

### Syntax

---

**comb -o -s [ -pSID ] [ -clist ] files**

### Description

---

The *comb* command generates a shell procedure (see *sh*(CP)) which, when run, will reconstruct the given SCCS files. The reconstructed files will, hopefully, be smaller than the original files. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *comb* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with *s*.) and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored. The generated shell procedure is written on the standard output.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed, but the effects of any keyletter argument apply independently to each named file.

- o** For each *get -e* generated, this argument causes the reconstructed file to be accessed at the release of the delta to be created, otherwise the reconstructed file would be accessed at the most recent ancestor. Use of the **-o** keyletter may decrease the size of the reconstructed SCCS file. It may also alter the shape of the delta tree of the original file.
- pSID** The SCCS identification string (SID) of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file. For example, if you give the command:

**comb -p 1.5 s.filename**

All deltas from 1.5 to the most recent delta will be preserved. All deltas prior to 1.5 will be removed.

- s** This argument causes *comb* to generate a shell procedure that, when run, produces a report giving the following for each file: the file name, the percentage change in file size after combining, the total file size (in blocks) after combining, and the original file size (also in blocks). The format of the report is as follows:

*filename* [+]*percentage\_change* *newsize/oldsize*

A sample command line to use this option would look like this:

```
comb -s s.filename | sh > report
```

where *report* is the name of the file that receives the report information.

It is recommended that before any SCCS files are actually combined, you should use this option to determine exactly how much space can be saved through the combining process.

#### **-clist**

This command causes the existing deltas in the subject file to be combined. It then overwrites the subject file with the new combined delta. For example, if you give the following command:

```
comb -c1.4 s.workfile | sh
```

all previous deltas to *s.workfile* will be combined, and the new combined delta will be number 1.4.

If no keyletter arguments are specified, *comb* will preserve only leaf deltas and the minimal number of ancestors needed to preserve the tree.

## **Files**

---

|           |                                          |
|-----------|------------------------------------------|
| s.COMB    | The name of the reconstructed SCCS file. |
| comb????? | Temporary.                               |

## **See Also**

---

admin(CP), delta(CP), get(CP), prs(CP), sccsfile(F) sh(C)

## **Notes**

---

The *comb* command may rearrange the shape of the tree of deltas. It may not save any space; in fact, it is possible for the reconstructed file to actually be larger than the original.



## conv

common object file converter

### Syntax

**conv** [-a] [-o] [-p] -t target [-l files]

### Description

The *conv* command converts object files in the common object file format from their current byte ordering to the byte ordering of the *target* machine. The converted file is written to *file.v*. The *conv* command can be used on either the source (sending) or target (receiving) machine.

Command line options are:

- Indicates that the names of *files* should be read from the standard input.
- a If the input file is an archive, produce the output file in the System V Release 2.0 portable archive format.
- o If the input file is an archive, produce the output file in the old (pre- System V) archive format.
- p If the input file is an archive, produce the output file in the System V Release 1.0 random access archive format.
- t target Convert the object file to the byte ordering of the machine (*target*) to which the object file is being shipped. This may be another host or a target machine. Legal values for *target* are: pdp, vax, ibm, x86, b16, n3b, mc68, and m32.

The *conv* command is meant to ease the problems created by a multi-host, cross-compilation development environment. The *conv* command is best used within a procedure for shipping object files from one machine to another.

The *conv* command will recognize and produce archive files in three formats: the pre- System V format, the System V Release 1.0 random access format, and the System V Release 2.0 portable ASCII format. By default, *conv* will create the output archive file in the same format as the input file. To produce an output file in a different format than the input file, use the -a, -o, or -p option. If the output archive format is the same as the input format, the archive symbol table will be

converted, otherwise the symbol table will be stripped from the archive. The *ar*(C) command with its *-t* and *-s* options must be used on the target machine to recreate the archive symbol table.

## Example

---

To ship object files from a VAX computer sytem to a 3B2 computer, execute the following commands:

```
conv -t m32 *.out
uucp *.out.v my3b2!~/rje/
```

## See Also

---

*ar*(CP), *convert*(CP), *ar*(F), *a.out*(F)

## Diagnostics

---

The diagnostics are self-explanatory. Fatal diagnostics on the command lines cause termination. Fatal diagnostics on an input file cause the program to continue to the next input file.

## Notes

---

The *conv* command will not convert archives from one format to another if both the source and target machines have the same byte ordering. The system tool *convert*(CP) should be used for this purpose.

## Standards Conformance

---

*conv* is conformant with:

AT&T SVID Issue 2, Select Code 307-127.



## convert

---

convert archive files to common formats

### Syntax

---

**convert** [-x] infile outfile

### Description

---

The *convert* command transforms input *infile* to output *outfile*. *infile* must be a System V Release 1.0 or XENIX archive file and *outfile* will be the equivalent System V Release 2.0 archive file. All other types of input to the *convert* command will be passed unmodified from the input file to the output file (along with appropriate warning messages).

The -x option is required to convert a XENIX archive. Using this option will convert the general archive structure but leave archive members unmodified.

*infile* must be different from *outfile*.

### Files

---

*TMPPDIR/conv\**

temporary files

*TMPPDIR* is usually */usr/tmp* but can be redefined by setting the environment variable **TMPPDIR** (see *tmpnam()* in *tmpnam(S)*).

### See Also

---

ar(CP), tmpnam(S) a.out(F), ar(F)

# cpp

## the C language preprocessor

### Syntax

**LIBDIR/cpp** [ option ... ] [ ifile [ ofile ] ]

### Description

The C language preprocessor, *cpp*, is invoked as the first pass of any C compilation by the *cc*(CP) command. Thus *cpp*'s output is designed to be in a form acceptable as input to the next pass of the C compiler. As the C language evolves, *cpp* and the rest of the C compilation package will be modified to follow these changes. Therefore, the use of *cpp* other than through the *cc*(CP) command is not suggested, since the functionality of *cpp* may someday be moved elsewhere. See *m4*(CP) for a general macro processor.

The *cpp* command optionally accepts two file names as arguments. *ifile* and *ofile* are, respectively, the input and output for the preprocessor. They default to standard input and standard output if not supplied.

The following *options* to *cpp* are recognized:

- P Preprocess the input without producing the line control information used by the next pass of the C compiler.
- C By default, *cpp* strips C-style comments. If the -C option is specified, all comments (except those found on *cpp* directive lines) are passed along.

#### -Uname

Remove any initial definition of *name*, where *name* is a reserved symbol that is predefined by the particular preprocessor. Following is the current list of these possibly reserved symbols. On the 80386, *unix* and *i386* are defined.

|                   |                                                                               |
|-------------------|-------------------------------------------------------------------------------|
| operating system: | unix, dmert, gcos, ibm, os, tss                                               |
| hardware:         | i286, i386, interdata, pdp11,<br>u370, u3b, u3b5, u3b2, u3b15,<br>u3b20d, vax |
| system variant:   | RES, RT                                                                       |
| lint(CP):         | lint                                                                          |



**-Dname****-Dname=def**

Define *name* with value *def* as if by a **#define**. If no *=def* is given, *name* is defined with value 1. The **-D** option has lower precedence than the **-U** option. That is, if the same name is used in both a **-U** option and a **-D** option, the name will be undefined regardless of the order of the options.

**-T** The **-T** option forces *cpp* to use only the first eight characters to distinguish preprocessor symbols and is included for backward compatibility.

**-Idir**

Change the algorithm for searching for **#include** files whose names do not begin with / to look in *dir* before looking in the directories on the standard list. Thus, **#include** files whose names are enclosed in " " will be searched for first in the directory of the file with the **#include** line, then in directories named in **-I** options, and last in directories on a standard list. For **#include** files whose names are enclosed in <>, the directory of the file with the **#include** line is not searched.

**-Ydir**

Use directory *dir* in place of the standard list of directories when searching for **#include** files.

**-H** Print, one per line on standard error, the path names of included files.

Two special names are understood by *cpp*. The name `__LINE__` is defined as the current line number (as a decimal integer) as known by *cpp*, and `__FILE__` is defined as the current file name (as a C string) as known by *cpp*. They can be used anywhere (including in macros) just as any other defined name.

All *cpp* directive lines start with # in column 1. Any number of blanks and tabs is allowed between the # and the directive. The directives are:

**#define name token-string**

Replace subsequent instances of *name* with *token-string*.

**#define name( arg, ..., arg ) token-string**

Notice that there can be no space between *name* and the (. Replace subsequent instances of *name* followed by a (, a list of comma-separated sets of tokens, and a ) followed with *token-string*. Each occurrence of an *arg* is replaced by the corresponding set of tokens in the comma-separated list. When a macro with arguments is expanded, the arguments are placed into the expanded *token-string* unchanged. After the entire *token-string* has been expanded, *cpp* re-starts its scan for names to expand at the beginning of the newly created *token-string*.

**#undef name**

Cause the definition of *name* (if any) to be forgotten from now on. No additional tokens are permitted on the directive line after *name*.

**#ident "string"**

Put *string* into the .comment section of an object file.

**#include "filename"****#include <filename>**

Include at this point the contents of *filename* (which will then be run through *cpp*). When the <*filename*> notation is used, *filename* is only searched for in the standard places. See the **-I** and **-Y** options above for more detail. No additional tokens are permitted on the directive line after the final " or >.

**#line integer-constant "filename"**

Causes *cpp* to generate line control information for the next pass of the C compiler. *Integer-constant* is the line number of the next line and *filename* is the file from which it comes. If "*filename*" is not given, the current file name is unchanged. No additional tokens are permitted on the directive line after the optional *filename*.

**#endif**

Ends a section of lines begun by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**. No additional tokens are permitted on the directive line.

**#ifdef name**

The lines following will appear in the output if and only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**. No additional tokens are permitted on the directive line after *name*.

**#ifndef name**

The lines following will appear in the output if and only if *name* has not been the subject of a previous **#define**. No additional tokens are permitted on the directive line after *name*.

**#if constant-expression**

Lines following will appear in the output if and only if the *constant-expression* evaluates to non-zero. All binary non-assignment C operators, the ?: operator, the unary -, !, and ~ operators are all legal in *constant-expression*. The precedence of the operators is the same as defined by the C language. There is also a unary operator **defined**, which can be used in *constant-expression* in these two forms: **defined** ( *name* ) or **defined** *name*. This allows the utility of **#ifdef** and **#ifndef** in a **#if** directive. Only these operators, integer constants, and names which are known by *cpp* should be used in *constant-expression*. In particular, the **sizeof** operator is not available.



To test whether either of two symbols, *foo* and *fum*, are defined, use

```
#if defined(foo) || defined(fum)
```

#### **#elif** *constant-expression*

An arbitrary number of **#elif** directives is allowed between a **#if**, **#ifdef**, or **#ifndef** directive and a **#else** or **#endif** directive. The lines following the **#elif** directive will appear in the output if and only if the preceding test directive evaluates to zero, all intervening **#elif** directives evaluate to zero, and the *constant-expression* evaluates to non-zero. If *constant-expression* evaluates to non-zero, all succeeding **#elif** and **#else** directives will be ignored. Any *constant-expression* allowed in a **#if** directive is allowed in a **#elif** directive.

#### **#else**

The lines following will appear in the output if and only if the preceding test directive evaluates to zero, and all intervening **#elif** directives evaluate to zero. No additional tokens are permitted on the directive line.

The test directives and the possible **#else** directives can be nested.

#### **#pragma pack**([1|2|4])

If an argument is present, subsequent structures will be aligned to the given byte boundary. The packing of structure members remains in effect until changed or disabled. If no argument is present and the **-Zp** option was used with the *cc* command, packing reverts to the packing specified on the *cc* command line. If the **-Zp** option was not used with the *cc* command, structures are aligned to their normal settings.

## Files

---

|               |                                                                                |
|---------------|--------------------------------------------------------------------------------|
| <i>INCDIR</i> | standard directory list for <b>#include</b> files, usually <i>/usr/include</i> |
| <i>LIBDIR</i> | usually <i>/lib</i>                                                            |

## See Also

---

*cc*(CP), *lint*(CP), *m4*(CP)

## Diagnostics

---

The error messages produced by *cpp* are intended to be self-explanatory. The line number and file name where the error occurred are printed along with the diagnostic.

## Notes

---

The unsupported **-W** option enables the **#class** directive. If it encounters a **#class** directive, *cpp* will exit with code 27 after finishing all other processing. This option provides support for "C with classes."

Because the standard directory for included files may be different in different environments, this form of **#include** directive:

```
#include <file.h>
```

should be used, rather than one with an absolute path, like:

```
#include "/usr/include/file.h"
```

The *cpp* command warns about the use of the absolute path name.

## Standards Conformance

---

*cpp* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



## **cprs**

---

compress a common object file

### **Syntax**

---

**cprs** [-p] file1 file2

### **Description**

---

The *cprs* command reduces the size of a common object file, *file1*, by removing duplicate structure and union descriptors. The reduced file, *file2*, is produced as output.

The sole option to *cprs* is:

**-p**     Print statistical messages including: total number of tags, total duplicate tags, and total reduction of *file1*.

### **Notes**

---

Note that only COFF files are supported by this utility.

### **See Also**

---

strip(CP), a.out(F), syms(F)

## cref

---

makes a cross-reference listing

### Syntax

---

**cref** [ **-acilnostux123** ] files

### Description

---

*cref* makes a cross-reference listing of assembler or C programs. The program searches the given *files* for symbols in the appropriate C or assembly language syntax.

The output report is in four columns:

1. Symbol
2. Filename
3. Current symbol or line number
4. Text as it appears in the file

*cref* uses either an *ignore* file or an *only* file. If the **-i** option is given, the next argument is taken to be an *ignore* file; if the **-o** option is given, the next argument is taken to be an *only* file. *ignore* and *only* files are lists of symbols separated by newlines. All symbols in an *ignore* file are ignored in columns 1 and 3 of the output. If an *only* file is given, only symbols in that file will appear in column 1. Only one of these options may be given; the default setting is **-i** using the default ignore file (see *FILES* below). Assembler predefined symbols or C keywords are ignored.

The **-s** option causes current symbols to be put in column 3. In the assembler, the current symbol is the most recent name symbol; in C, the current function name. The **-l** option causes the line number within the file to be put in column 3.

The **-t** option causes the next available argument to be used as the name of the intermediate file (instead of the temporary file */tmp/crt??*). This file is created and is *not* removed at the end of the process.



The *cref* options are:

- a Uses assembler format (default)
- c Uses C format
- i Uses an *ignore* file (see above)
- l Puts line number in column 3 (instead of current symbol)
- n Omits column 4 (no context)
- o Uses an *only* file (see above)
- s Current symbol in column 3 (default)
- t User-supplied temporary file
- u Prints only symbols that occur exactly once
- x Prints only C external symbols
- 1 Sorts output on column 1 (default)
- 2 Sorts output on column 2
- 3 Sorts output on column 3

## Files

---

/usr/lib/cref/\* Assembler specific files

## See Also

---

as(CP), cc(CP), sort(C), xref(CP)

## Notes

---

*cref* inserts an ASCII DEL character into the intermediate file after the eighth character of each name that is eight or more characters long in the source file.

## cscope

interactively examine a C program

### Syntax

```
cscope [-f reffile] [-i namefile] [-I incdir] [-d] [files]
```

### Description

*cscope* is an interactive screen-oriented tool that helps programmers browse through C source code.

By default, *cscope* examines the C, *yacc*, and *lex* source files in the current directory and builds a symbol cross-reference. It then uses this table to find references to symbols (including C preprocessor symbols), function declarations, and function calls.

*cscope* builds the symbol cross-reference the first time it is used on the source files for the program being browsed. On a subsequent invocation, *cscope* rebuilds the cross-reference only if a source file has changed or the list of source files is different. When the cross-reference is rebuilt, the data for the unchanged files are copied from the old cross-reference, which makes rebuilding much faster than the initial build.

The following options can appear in any combination:

**-f *reffile***

Use *reffile* as the cross-reference file name instead of the default *cscope.out*.

**-i *namefile***

Get the list of files (file names separated by spaces, tabs, or new-lines) to browse from *namefile*. If this option is specified, *cscope* ignores any files appearing on the command line.

**-I *incdir***

Look in *incdir* (before looking in **INCDIR**, the standard place for header files that is normally **/usr/include**) for any **#include** files whose names do not begin with **/** and that are not specified on the command line or in *namefile* above. (The **#include** files may be specified with either double quotes or angle brackets.) The *incdir* directory is searched in addition to the current directory (which is searched first) and the standard list (which is searched last). If more than one occurrence of **-I** appears, the directories are searched in the order they appear on the command line.

**-d** Do not update the cross-reference.



## Requesting the Initial Search

After the cross-reference is ready *cscope* will display this menu:

```
List references to this C symbol:
Edit this function or #define:
List functions called by this function:
List functions calling this function:
List lines containing this text string:
Change this text string:
```

Press the **TAB** key repeatedly to move to the desired input field, type the text to search for, and then press the **RETURN** key.

## Issuing Subsequent Requests

If the search is successful, any of these single-character commands can be used:

|              |                                                    |
|--------------|----------------------------------------------------|
| <b>1-9</b>   | Edit the file referenced by the given line number. |
| <b>SPACE</b> | Display next lines.                                |
| <b>+</b>     | Display next lines.                                |
| <b>-</b>     | Display previous lines.                            |
| <b>^e</b>    | Edit all lines.                                    |
| <b>&gt;</b>  | Append the displayed list of lines to a file.      |

At any time these single-character commands can also be used:

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <b>TAB</b>    | Move to next input field.                                                |
| <b>RETURN</b> | Move to next input field.                                                |
| <b>^m</b>     | Move to next input field.                                                |
| <b>^p</b>     | Move to previous input field.                                            |
| <b>.</b>      | Search with the last text typed.                                         |
| <b>^r</b>     | Rebuild the cross-reference.                                             |
| <b>!</b>      | Start an interactive shell (type <b>^d</b> to return to <i>cscope</i> ). |
| <b>^l</b>     | Redraw the screen.                                                       |
| <b>?</b>      | Display this list of commands.                                           |
| <b>^d</b>     | Exit <i>cscope</i> .                                                     |

Note: If the first character of the text to be searched for matches one of the above commands, escape it by typing a **\** (backslash) first.

## Substituting New Text for Old Text

After the text to be changed has been typed, *cscope* will prompt for the new text, and then it will display the lines containing the old text. Select the lines to be changed with these single-character commands:

- 1-9**        Mark or unmark the line to be changed.
- \***         Mark or unmark all displayed lines to be changed.
- SPACE**    Display next lines.
- +**         Display next lines.
- Display previous lines.
- a**         Mark all lines to be changed.
- ^d**        Change the marked lines and exit.
- ESCAPE**   Exit without changing the marked lines.
- !**         Start an interactive shell (type `<CTL>-D` to return to `cscope`).
- ^L**        Redraw the screen.
- ?**         Display this list of commands.

## Environment Variables

---

- EDITOR**    Preferred editor, which defaults to `vi(C)`.
- HOME**     Home directory, which is automatically set at login.
- SHELL**    Preferred shell, which defaults to `sh(C)`.
- TERM**     Terminal type, which must be a screen terminal.
- VIEWER**   Preferred file display program (such as `pg(C)`), which overrides **EDITOR** (see above).
- VPATH**    An ordered list of directory names, separated by colons. It can be used by `cscope` to search for both source and header files, but the two types of files have different orders of search. If **VPATH** is set, `cscope` searches for source files in the directories specified; if it is not set, `cscope` searches only in the current directory. `cscope` searches for header files in the following order: (C) if **VPATH** is set, in directories specified in **VPATH** and if **VPATH** is not set, in the current directory; (S) in directories specified by the **-I** option (if they exist); and (S) in the standard location for header files (normally `/usr/include`).



## Files

---

|                    |                                                                                                                   |
|--------------------|-------------------------------------------------------------------------------------------------------------------|
| <b>cscope.out</b>  | Symbol cross-reference file, which is put in the home directory if it cannot be created in the current directory. |
| <b>ncscope.out</b> | Temporary file containing new cross-reference before it replaces the old cross-reference.                         |
| <b>INCDIR</b>      | Standard directory for <b>#include</b> files (usually is <b>/usr/include</b> ).                                   |

## Warnings

---

*cscope* recognizes function definitions of the form:

*fname blank ( args ) white arg\_decs white {*

where:

|                 |                                                                                               |
|-----------------|-----------------------------------------------------------------------------------------------|
| <i>fname</i>    | is the function name,                                                                         |
| <i>blank</i>    | is zero or more spaces or tabs, not including newlines,                                       |
| <i>args</i>     | is any string that does not contain a " or a newline,                                         |
| <i>white</i>    | is zero or more spaces, tabs, or newlines, and                                                |
| <i>arg_decs</i> | are zero or more argument declarations. <i>arg_decs</i> may include comments and white space. |

It is not necessary for a function declaration to start at the beginning of a line. The return type may precede the function name; *cscope* will still recognize the declaration. Function definitions that deviate from this form will not be recognized by *cscope*.

## ctags

---

create a tags file

### Syntax

---

```
ctags [-a] [-u] [-v] [-w] [-x] name ...
```

### Description

---

*ctags* makes a tags file for *vi*(C) from the specified C sources. A tags file gives the locations of specified objects (in this case functions) in a group of files. Each line of the tags file contains the function name, the file in which it is defined, and a scanning pattern used to find the function definition. These are given in separate fields on the line, separated by blanks or tabs. Using the tags file, *vi* can quickly find these function definitions.

If the *-x* flag is given, *ctags* produces a list of function names and the line number and file name on which each is defined, as well as the text of that line, and prints this on the standard output. With the *-x* option, no tags file is created. This is a simple index which can be printed out as an off-line readable function index. Files whose name ends in *.c* or *.h* are assumed to be C source files and are searched for C routine and macro definitions.

Other options are

- a** Causes specified files to be appended to tags; that is, new values for the files are appended to the tags file.
- u** Causes the specified files to be *updated* in tags; that is, all references to them are deleted, and the new values are appended to the file. (Beware: this option is implemented in a way which is rather slow; it is usually faster to simply rebuild the tags file.)
- v** Produces a list of function names, the filename each function is declared in, and the function's line number. This list prints on the standard output, and no tags file is created.
- w** Suppresses warning diagnostics.

The tag *main* is treated specially in C programs. The tag formed is created by prefixing **M** to the name of the file, with a trailing *.c* removed, if any, and leading path name components also removed. This makes use of *ctags* practical in directories with more than one program.



## **Files**

---

tags

Output tags file

## **See Also**

---

ex(C), vi(C)

## **Credit**

---

This utility was developed at the University of California at Berkeley and is used with permission.

# ctrace

## C program debugger

---

### Syntax

---

**ctrace** [options] [file]

### Description

---

The *ctrace* command allows you to follow the execution of a C program, statement-by-statement. The effect is similar to executing a shell procedure with the *-x* option. The *ctrace* command reads the C program in *file* (or from standard input if you do not specify *file*), inserts statements to print the text of each executable statement and the values of all variables referenced or modified, and writes the modified program to the standard output. You must put the output of *ctrace* into a temporary file because the *cc*(CP) command does not allow the use of a pipe. You then compile and execute this file.

As each statement in the program executes it will be listed at the terminal, followed by the name and value of any variables referenced or modified in the statement, followed by any output from the statement. Loops in the trace output are detected and tracing is stopped until the loop is exited or a different sequence of statements within the loop is executed. A warning message is printed every 1000 times through the loop to help you detect infinite loops. The trace output goes to the standard output so you can put it into a file for examination with an editor or the *bfs*(C) or *tail*(C) commands.

The options commonly used are:

- f functions**      Trace only these *functions*.
- v functions**      Trace all but these *functions*.

You may want to add to the default formats for printing variables. Long and pointer variables are always printed as signed integers. Pointers to character arrays are also printed as strings if appropriate. Char, short, and int variables are also printed as signed integers and, if appropriate, as characters. Double variables are printed as floating point numbers in scientific notation. You can request that variables be printed in additional formats, if appropriate, with these options:



- o Octal
- x Hexadecimal
- u Unsigned
- e Floating point

These options are used only in special circumstances:

- l *n* Check *n* consecutively executed statements for looping trace output, instead of the default of 20. Use 0 to get all the trace output from loops.
- s Suppress redundant trace output from simple assignment statements and string copy function calls. This option can hide a bug caused by use of the = operator in place of the == operator.
- t *n* Trace *n* variables per statement instead of the default of 10 (the maximum number is 20). The Diagnostics section explains when to use this option.
- P Run the C preprocessor on the input before tracing it. You can also use the -D, -I, and -U *cpp*(CP) options.

These options are used to tailor the run-time trace package when the traced program will run in a non-UNIX type environment:

- b Use only basic functions in the trace code, that is, those in *ctype*(S), *printf*(S), and *string*(S). These are usually available even in cross-compilers for microprocessors. In particular, this option is needed when the traced program runs under an operating system that does not have *signal*(S), *flush*(S), *longjmp*(S), or *setjmp*(S).
- p *string* Change the trace print function from the default of 'printf'. For example, 'fprintf(stderr, ' would send the trace to the standard error output.
- r *f* Use file *f* in place of the *runtime.c* trace function package. This lets you change the entire print function, instead of just the name and leading arguments (see the -p option).

## Example

---

If the file *lc.c* contains this C program:

```
1 #include <stdio.h>
2 main() /* count lines in input */
3 {
4 int c, nl;
5
6 nl = 0;
7 while ((c = getchar()) != EOF)
8 if (c == '\n')
9 ++nl;
10 printf("%d\n", nl);
11 }
```

and you enter these commands and test data:

```
cc lc.c
a.out
1
(cntl-d)
```

the program will be compiled and executed. The output of the program will be the number **2**, which is not correct because there is only one line in the test data. The error in this program is common, but subtle. If you invoke *ctrace* with these commands:

```
ctrace lc.c >temp.c
cc temp.c
a.out
```

the output will be:

```
2 main()
6 nl = 0;
 /* nl == 0 */
7 while ((c = getchar()) != EOF)
```

The program is now waiting for input. If you enter the same test data as before, the output will be:



```

 /* c == 49 or '1' or "text" */
8 if (c == '\n')
 /* c == 10 or '\n' */
9 ++nl;
 /* nl == 1 */
7 while ((c = getchar()) != EOF)
 /* c == 10 or '\n' */
8 if (c == '\n')
 /* c == 10 or '\n' */
9 ++nl;
 /* nl == 2 */
7 /* repeating */

```

If you now enter an end-of-file character (cntl-d) the final output will be:

```

 /* repeated < 1 time */
7 while ((c = getchar()) != EOF)
 /* c == -1 */
10 printf ("%d0, nl);
 /* nl == 2 */2
 /* return */

```

Note that the program output printed at the end of the trace line for the **nl** variable. Also note the **return** comment added by *ctrace* at the end of the trace output. This shows the implicit return at the terminating brace in the function.

The trace output shows that variable **c** is assigned the value '1' in line 7, but in line 8 it has the value '\n'. Once your attention is drawn to this **if** statement, you will probably realize that you used the assignment operator (=) in place of the equality operator (==). You can easily miss this error during code reading.

## Execution-Time Trace Control

The default operation for *ctrace* is to trace the entire program file, unless you use the **-f** or **-v** options to trace specific functions. This does not give you statement-by-statement control of the tracing, nor does it let you turn the tracing off and on when executing the traced program.

You can do both of these by adding *ctroff()* and *ctron()* function calls to your program to turn the tracing off and on, respectively, at execution time. Thus, you can code arbitrarily complex criteria for trace control with *if* statements, and you can even conditionally include this code because *ctrace* defines the **CTRACE** preprocessor variable. For example:

```

#ifdef CTRACE
 if (c == '!' && i > 1000)
 ctron();
#endif

```

You can also call these functions from *sdb*(CP) if you compile with the **-g** option. For example, to trace all but lines 7 to 10 in the main function, enter:

```

sdb a.out
main:7b ctroff()
main:11b ctron()
r

```

You can also turn the trace off and on by setting static variable *tr\_ct\_* to 0 and 1, respectively. This is useful if you are using a debugger that cannot call these functions directly.

## Files

---

/usr/lib/ctrace/runtime.c

run-time trace package

## See Also

---

signal(S), ctype(S), fclose(S), printf(S), setjmp(S), string(S), bfs(C), tail(C)

## Diagnostics

---

This section contains diagnostic messages from both *ctrace* and *cc*(CP), since the traced code often gets some *cc* warning messages. You can get *cc* error messages in some rare cases, all of which can be avoided.

### ctrace Diagnostics

*warning: some variables are not traced in this statement*

Only 10 variables are traced in a statement to prevent the C compiler "out of tree space; simplify expression" error. Use the **-t** option to increase this number.

*warning: statement too long to trace*

This statement is over 400 characters long. Make sure that you are using tabs to indent your code, not spaces.

*cannot handle preprocessor code, use -P option*

This is usually caused by *#ifdef*/*#endif* preprocessor statements in the middle of a C statement, or by a semicolon at the end of a *#define* preprocessor statement.



*'if ... else if' sequence too long*

Split the sequence by removing an **else** from the middle.

*possible syntax error, try -P option*

Use the **-P** option to preprocess the *ctrace* input, along with any appropriate **-D**, **-I**, and **-U** preprocessor options. If you still get the error message, check the Warnings section below.

## cc Diagnostics

*warning: illegal combination of pointer and integer*

*warning: statement not reached*

*warning: sizeof returns 0*

Ignore these messages.

*compiler takes size of function*

See the *ctrace* "possible syntax error" message above.

*yacc stack overflow*

See the *ctrace* "'if ... else if' sequence too long" message above.

*out of tree space; simplify expression*

Use the **-t** option to reduce the number of traced variables per statement from the default of 10. Ignore the "ctrace: too many variables to trace" warnings you will now get.

*redeclaration of signal*

Either correct this declaration of *signal(S)*, or remove it and `#include <signal.h>`.

## Warnings

---

You will get a *ctrace* syntax error if you omit the semicolon at the end of the last element declaration in a structure or union, just before the right brace (}). This is optional in some C compilers.

Defining a function with the same name as a system function may cause a syntax error if the number of arguments is changed. Just use a different name.

The *ctrace* command assumes that **BADMAG** is a preprocessor macro, and that **EOF** and **NULL** are `#defined` constants. Declaring any of these to be variables, for example, `"int EOF;"`, will cause a syntax error.

## Notes

---

The *ctrace* command does not know about the components of aggregates like structures, unions, and arrays. It cannot choose a format to print all the components of an aggregate when an assignment is made to the entire aggregate. *ctrace* may choose to print the address of an aggregate or use the wrong format (for example, 3.149050e-311 for a structure with two integer members) when printing the value of an aggregate.

Pointer values are always treated as pointers to character strings.

The loop trace output elimination is done separately for each file of a multifile program. This can result in functions called from a loop still being traced, or the elimination of trace output from one function in a file until another in the same file is called.



## cxref

generate C program cross-reference

### Syntax

**cxref** [ options ] files

### Description

The *cxref* command analyzes a collection of C files and attempts to build a cross-reference table. The *cxref* command uses a special version of *cpp* to include **#define**'d information in its symbol table. It produces a listing on standard output of all symbols (auto, static, and global) in each file separately, or, with the **-c** option, in combination. Each symbol contains an asterisk (\*) before the declaring reference.

In addition to the **-D**, **-I**, and **-U** options (which are interpreted just as they are by *cc*(CP) and *cpp*(CP)), the following *options* are interpreted by *cxref*:

- c**            Print a combined cross-reference of all input files.
- w<num>**      Width option which formats output no wider than *<num>* (decimal) columns. This option will default to 80 if *<num>* is not specified or is less than 51.
- o file**        Direct output to *file*.
- s**            Operate silently; do not print input file names.
- t**            Format listing for 80-column width.

### Files

- LLIBDIR*            usually /usr/lib
- LLIBDIR/xcpp*      special version of the C preprocessor.

### See Also

*cc*(CP), *cpp*(CP)

## Diagnostics

---

Error messages are unusually cryptic, but usually mean that you cannot compile these files.

## Notes

---

The *cxref* command considers a formal argument in a *#define* macro definition to be a declaration of that symbol. For example, a program that *#includes ctype.h*, will contain many declarations of the variable *c*.

## Standards Conformance

---

*cxref* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



## delta

---

make a delta (change) to an SCCS file

### Syntax

---

**delta** [-rSID] [-s] [-n] [-glist] [-m[mrlist]] [-y[comment]] [-p] files

### Description

---

The *delta* command is used to permanently introduce into the named SCCS file changes that were made to the file retrieved by *get*(CP) (called the *g-file*, or generated file).

The *delta* command makes a delta to each named SCCS file. If a directory is named, *delta* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with *s*.) and unreadable files are silently ignored. If a name of - is given, the standard input is read (see *Warnings*); each line of the standard input is taken to be the name of an SCCS file to be processed.

The *delta* command may issue prompts on the standard output depending upon certain keyletters specified and flags (see *admin*(CP)) that may be present in the SCCS file (see -m and -y keyletters below).

Keyletter arguments apply independently to each named file.

- |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-rSID</b> | Uniquely identifies which delta is to be made to the SCCS file. The use of this keyletter is necessary only if two or more outstanding <i>gets</i> for editing ( <b>get -e</b> ) on the same SCCS file were done by the same person (login name). The SID value specified with the -r keyletter can be either the SID specified on the <i>get</i> command line or the SID to be made as reported by the <i>get</i> command (see <i>get</i> (CP)). A diagnostic results if the specified SID is ambiguous or if a required SID is omitted on the command line. |
| <b>-s</b>    | Suppresses the issue, on the standard output, of the created delta's SID, as well as the number of lines inserted, deleted, and unchanged in the SCCS file.                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>-n</b>    | Specifies retention of the edited <i>g-file</i> (normally removed at completion of delta processing).                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

**-glist**

a *list* (see *get*(CP) for the definition of *list*) of deltas which are to be *ignored* when the file is accessed at the change level (SID) created by this delta.

**-m[mrlist]**

If the SCCS file has the **v** flag set (see *admin*(CP)), then a Modification Request (**MR**) number *must* be supplied as the reason for creating the new delta.

If **-m** is not used and the standard input is a terminal, the prompt **MRs?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The **MRs?** prompt always precedes the **comments?** prompt (see **-y** keyletter).

**MRs** in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the **MR** list.

Note that if the **v** flag has a value (see *admin*(CP)), it is taken to be the name of a program (or shell procedure) which will validate the correctness of the **MR** numbers. If a non-zero exit status is returned from the **MR** number validation program, *delta* terminates. (It is assumed that the **MR** numbers were not all valid.)

**-y[comment]**

Arbitrary text used to describe the reason for making the delta. A null string is considered a valid *comment*.

If **-y** is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the comment text.

**-p**

Causes *delta* to print (on the standard output) the SCCS file differences before and after the delta is applied in a *diff*(C) format.



## Files

---

|                |                                                                                                        |
|----------------|--------------------------------------------------------------------------------------------------------|
| g-file         | Existed before the execution of <i>delta</i> ; removed after completion of <i>delta</i> .              |
| p-file         | Existed before the execution of <i>delta</i> ; may exist after completion of <i>delta</i> .            |
| q-file         | Created during the execution of <i>delta</i> ; removed after completion of <i>delta</i> .              |
| x-file         | Created during the execution of <i>delta</i> ; renamed to SCCS file after completion of <i>delta</i> . |
| z-file         | Created during the execution of <i>delta</i> ; removed during the execution of <i>delta</i> .          |
| d-file         | Created during the execution of <i>delta</i> ; removed after completion of <i>delta</i> .              |
| /usr/bin/bdiff | Program to compute differences between the "gotten" file and the <i>g-file</i> .                       |

## See Also

---

admin(CP), cdc(CP), get(CP), prs(CP), rmdel(CP), sccsfile(F), bdiff(C)

## Warnings

---

Lines beginning with an SOH ASCII character (binary 001) cannot be placed in the SCCS file unless the SOH is escaped. This character has special meaning to SCCS (see *sccsfile(F)*) and will cause an error.

A *get* of many SCCS files, followed by a *delta* of those files, should be avoided when the *get* generates a large amount of data. Instead, multiple *get/delta* sequences should be used.

If the standard input (-) is specified on the *delta* command line, the **-m** (if necessary) and **-y** keyletters *must* also be present. Omission of these keyletters causes an error to occur.

Comments are limited to text strings of at most 512 characters.

## Standards Conformance

---

*delta* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## dis

object code disassembler

### Syntax

**dis** [-o] [-V] [-L] [-s] [-d sec] [-da sec] [-F function] [-t sec]  
[-l string] file ...

### Description

The *dis* command produces an assembly language listing of *file*, which may be an object file or an archive of object files. The listing includes assembly statements and an octal or hexadecimal representation of the binary that produced those statements.

The following *options* are interpreted by the disassembler and may be specified in any order.

- o** Print numbers in octal. The default is hexadecimal.
- V** Print, on standard error, the version number of the disassembler being executed.
- L** Look up source labels in the symbol table for subsequent printing. This option works only if the file was compiled with additional debugging information (for example, the **-g** option of *cc*(CP)).
- s** Perform symbolic disassembly, that is, specify source symbol names for operands where possible. Symbolic disassembly output will appear on the line following the instruction. For maximal symbolic disassembly to be performed, the file must be compiled with additional debugging information (for example, the **-g** option of *cc*(CP)). Symbol names will be printed using C syntax.
- d sec** Disassemble the named section as data, printing the offset of the data from the beginning of the section.
- da sec** Disassemble the named section as data, printing the actual address of the data.
- F function** Disassemble only the named function in each object file specified on the command line. The **-F** option may be specified multiple times on the command line.



- t** *sec*            Disassemble the named section as text.
- l** *string*        Disassemble the library file specified by *string*. For example, one would issue the command **dis -l x -l z** to disassemble **libx.a** and **libz.a**. All libraries are assumed to be in *LIBDIR*.

If the **-d**, **-da** or **-t** options are specified, only those named sections from each user-supplied file name will be disassembled. Otherwise, all sections containing text will be disassembled.

On output, a number enclosed in brackets at the beginning of a line, such as [5], represents that the break-pointable line number starts with the following instruction. These line numbers will be printed only if the file was compiled with additional debugging information (for example, the **-g** option of *cc*(CP)). An expression such as <40> in the operand field or in the symbolic disassembly, following a relative displacement for control transfer instructions, is the computed address within the section to which control will be transferred. A function name will appear in the first column, followed by ().

## Notes

---

Note that this utility operates only on COFF format executables.

## Files

---

*LIBDIR*            usually /lib.

## See Also

---

*as*(CP), *cc*(CP), *ld*(CP), *a.out*(F)

## Diagnostics

---

The self-explanatory diagnostics indicate errors in the command line or problems encountered with the specified files.

## Standards Conformance

---

*dis* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

# dosld

---

## MS-DOS cross linker

---

### Syntax

---

**dosld** [ options ] file ...

---

### Description

---

*dosld* links the object files(s) given by *file* to create a program for execution under MS-DOS. Although similar to *ld*(CP), *dosld* has many options that differ significantly from *ld*. The options are described below:

- D** DS Allocate. This instructs *dosld* to perform DS allocation. It is generally used in conjunction with the **-H** option.
- H** Load high. This option instructs *dosld* to set a field in the header of the executable file to tell MS-DOS to load the program at the highest available position in memory. It is most often used with programs in which data precedes code in the memory image.
- L** Include line numbers. This option instructs *dosld* to include line numbers in the listing file (if any). Note that *dosld* cannot put line numbers in the listing file if the source translator hasn't put them in the object file.
- M**  
Include public symbols. This option instructs *dosld* to include public symbols in the list file. The symbols are sorted twice, lexicographically and by address.
- C** Ignore case. This option instructs *dosld* to treat upper and lower case characters in symbol names as identical.
- F** *num*  
Set stack size. This option should be followed by a hexadecimal number. *dosld* will use this number for the size in bytes of the stack segment in the output file.
- S** *num*  
Set segment limit. This option should be followed by a decimal number between 1 and 1024. The number sets the limit on the number of different segments that may be linked together. The default is 128. Note that the higher the value given, the slower the link will be.



**-m filename**

Create map file. This option should be followed by a filename. *dosld* will create a file with the given name in which it will put information about the segments and groups in the executable. Additionally, public symbols and line numbers will be listed in this file if the **-M** and **-L** options are given.

**-nl num**

Set name length. This option should be followed by a decimal number. The option instructs *dosld* to truncate all public and external symbols longer than *num* characters.

**-o filename**

Name output file. This option should be followed by a filename which *dosld* will use as the name of the executable file it creates. The default name is **a.out**.

**-u name**

Name undefined symbol. This option should be followed by a symbol name. *dosld* will enter the given name into its symbol table as an undefined symbol. The **-u** option may appear more than once on the command line.

**-G** Ignore group associations. This option instructs *dosld* to ignore any group definitions it may find in the input files. This option is provided for compatibility with old versions of MS-LINK; generally, it should never be used.

As with *ld*, the files passed to *dosld* may be either UNIX-style libraries (objects collected using *ar*(CP) and indexed using *ranlib*(CP)) or ordinary 8086 object files. Unless the **-u** option appears, at least one of the files passed to *dosld* must be an ordinary object file. Libraries are searched only after all the ordinary object files have been processed.

## Files

---

/usr/bin/dosld

## See Also

---

*ar*(CP), *as*(CP), *cc*(CP), *ld*(CP), *ranlib*(CP)

## Value Added

---

*dosld* is an extension of AT&T System V provided by the Santa Cruz Operation.

# dump

---

dump selected parts of an object file

---

## Syntax

**dump** [ options ] files

---

## Description

The *dump* command outputs selected parts of each of its object *file* arguments. The output can be directed via the standard redirection tools.

This command will accept both object files and archives of object files. It processes each file argument according to one or more of the following options:

- a**            Dump the archive header of each member of each archive file argument.
- g**            Dump the global symbols in the symbol table of an archive.
- f**            Dump each file header.
- o**            Dump each optional header.
- h**            Dump section headers.
- s**            Dump section contents.
- r**            Dump relocation information.
- l**            Dump line number information.
- t**            Dump symbol table entries.
- z name**       Dump line number entries for the named function.
- c**            Dump the string table.
- L**            Interpret and print the contents of the *.lib* sections.

The following *modifiers* are used in conjunction with the options listed above to modify their capabilities.



- d number** Dump the section number, *number*, or the range of sections starting at *number* and ending at the *number* specified by **+d**.
- +d number** Dump sections in the range either beginning with first section or beginning with section specified by **-d**.
- n name** Dump information pertaining only to the named entity. This *modifier* applies to **-h**, **-s**, **-r**, **-l**, and **-t**.
- p** Suppress printing of the headers.
- t index** Dump only the indexed symbol table entry. The **-t** used in conjunction with **+t**, specifies a range of symbol table entries.
- +t index** Dump the symbol table entries in the range ending with the indexed entry. The range begins at the first symbol table entry or at the entry specified by the **-t** option.
- u** Underline the name of the file for emphasis.
- v** Dump information in symbolic representation rather than numeric (for example, `C_STATIC` instead of `0X02`). This *modifier* can be used with all the above options except **-s** and **-o** options of *dump*.
- z name,number**  
Dump line number entry or range of line numbers starting at *number* for the named function.
- +z number** Dump line numbers starting at either function *name* or *number* specified by **-z**, up to *number* specified by **+z**.

Blanks separating an *option* and its *modifier* are optional. The comma separating the name from the number modifying the **-z** option may be replaced by a blank.

The *dump* command attempts to format the information it dumps in a meaningful way, printing certain information in character, hex, octal, or decimal representation as appropriate.

## Notes

---

Note that this utility operates only on COFF executables.

## See Also

---

a.out(F), ar(F).

## gencc

---

create a front-end to the cc command

### Syntax

---

`gencc`

### Description

---

The *gencc* command is an interactive command designed to aid in the creation of a front-end to the *cc* command. Since hard-coded path names have been eliminated from the C Compilation System (CCS), it is possible to move pieces of the CCS to new locations without recompiling the CCS. The new locations of moved pieces can be specified through the *-Y* option to the *cc* command. However, it is inconvenient to supply the proper *-Y* options with every invocation of the *cc* command. Further, if a system administrator moves pieces of the CCS, such movement should be invisible to users.

The front-end to the *cc* command which *gencc* generates is a one-line shell script which calls the *cc* command with the proper *-Y* options specified. The front-end to the *cc* command will also pass all user supplied options to the *cc* command.

The *gencc* command prompts for the location of each tool and directory which can be respecified by a *-Y* option to the *cc* command. If no location is specified, it assumes that that piece of the CCS has not been relocated. After all the locations have been prompted for, *gencc* will create the front-end to the *cc* command.

The *gencc* command creates the front-end to the *cc* command in the current working directory and gives the file the same name as the *cc* command. Thus, *gencc* can not be run in the same directory containing the actual *cc* command. Further, if a system administrator has redistributed the CCS, the actual *cc* command should be placed somewhere which is not typically in a user's PATH (for example, /lib). This will prevent users from accidentally invoking the *cc* command without using the front-end.

### Files

---

|                   |                 |
|-------------------|-----------------|
| <code>./cc</code> | front-end to cc |
|-------------------|-----------------|



## See Also

---

cc(CP)

## Notes

---

The *gencc* command does not produce any warnings if a tool or directory does not exist at the specified location. Also, *gencc* does not actually move any files to new locations.

# get

---

get a version of an SCCS file

## Syntax

---

**get** [-rSID] [-ccutoff] [-ilist] [-xlist] [-wstring] [-aseq-no.] [-k] [-e] [-l[p]] [-p] [-m] [-n] [-s] [-b] [-g] [-t] file ...

## Description

---

The *get* command generates an ASCII text file from each named SCCS file according to the specifications given by its keyletter arguments, which begin with -. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *get* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The generated text is normally written into a file called the *g-file* whose name is derived from the SCCS file name by simply removing the leading s.; (see also *Files*, below).

Each of the keyletter arguments is explained below as though only one SCCS file is to be processed, but the effects of any keyletter argument applies independently to each named file.

**-rSID** The SCCS identification string (SID) of the version (delta) of an SCCS file to be retrieved. Table 1 below shows, for the most useful cases, what version of an SCCS file is retrieved [as well as the SID of the version to be eventually created by *delta*(CP) if the -e keyletter is also used], as a function of the SID specified.

**-ccutoff** *Cutoff* date-time, in the form:

YY[MM[DD[HH[MM[SS]]]]]

No changes (deltas) to the SCCS file which were created after the specified *cutoff* date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, -c7502 is equivalent to -c750228235959. Any number of non-numeric characters may separate the



various 2-digit pieces of the *cutoff* date-time. This feature allows one to specify a *cutoff* date in the form: "-c77/2/2 9:22:25". Note that this implies that one may use the %E% and %U% identification keywords (see below) for nested *gets*.

get "-c%E% %U%" s.file

**-ilist** A *list* of deltas to be included (forced to be applied) in the creation of the generated file. The *list* has the following syntax:

<list> ::= <range> | <list> , <range>  
<range> ::= SID | SID - SID

SID, the SCCS Identification of a delta, may be in any form shown in the "SID Specified" column of Table 1.

**-xlist** A *list* of deltas to be excluded in the creation of the generated file. See the **-i** keyletter for the *list* format.

**-e** Indicates that the *get* is for the purpose of editing or making a change (delta) to the SCCS file via a subsequent use of *delta*(CP). The **-e** keyletter used in a *get* for a particular version (SID) of the SCCS file prevents further *gets* for editing on the same SID until *delta* is executed or the **j** (joint edit) flag is set in the SCCS file (see *admin*(CP)). Concurrent use of *get* **-e** for different SIDs is always allowed.

If the *g-file* generated by *get* with an **-e** keyletter is accidentally ruined in the process of editing it, it may be regenerated by re-executing the *get* command with the **-k** keyletter in place of the **-e** keyletter.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file (see *admin*(CP)) is enforced when the **-e** keyletter is used.

**-b** Used with the **-e** keyletter to indicate that the new delta should have an SID in a new branch as shown in Table 1. This keyletter is ignored if the **b** flag is not present in the file (see *admin*(CP)) or if the retrieved *delta* is not a leaf *delta*. (A leaf *delta* is one that has no successors on the SCCS file tree.)

Note: A branch *delta* may always be created from a non-leaf *delta*. Partial SIDs are interpreted as shown in the "SID Retrieved" column of Table 1.

- k** Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The **-k** keyletter is implied by the **-e** keyletter.
- l[p]** Causes a delta summary to be written into an *l-file*. If **-lp** is used, then an *l-file* is not created; the delta summary is written on the standard output instead. See *Files* for the format of the *l-file*.
- p** Causes the text retrieved from the SCCS file to be written on the standard output. No *g-file* is created. All output which normally goes to the standard output goes to file descriptor 2 instead, unless the **-s** keyletter is used, in which case it disappears.
- s** Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected.
- m** Causes each text line retrieved from the SCCS file to be preceded by the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.
- n** Causes each generated text line to be preceded with the *%M%* identification keyword value (see below). The format is: *%M%* value, followed by a horizontal tab, followed by the text line. When both the **-m** and **-n** keyletters are used, the format is: *%M%* value, followed by a horizontal tab, followed by the **-m** keyletter generated format.
- g** Suppresses the actual retrieval of text from the SCCS file. It is primarily used to generate an *l-file*, or to verify the existence of a particular SID.
- t** Used to access the most recently created delta in a given release (e.g., **-r1**), or release and level (e.g., **-r1.2**).
- w string** Substitute *string* for all occurrences of *%W%* when getting the file.



**-aseq-no.** The delta sequence number of the SCCS file delta (version) to be retrieved (see *sccsfile(F)*). This keyletter is used by the *comb*(CP) command; it is not a generally useful keyletter. If both the **-r** and **-a** keyletters are specified, only the **-a** keyletter is used. Care should be taken when using the **-a** keyletter in conjunction with the **-e** keyletter, as the SID of the delta to be created may not be what one expects. The **-r** keyletter can be used with the **-a** and **-e** keyletters to control the naming of the SID of the delta to be created.

For each file processed, *get* responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the **-e** keyletter is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a new-line) before it is processed. If the **-i** keyletter is used, included deltas are listed following the notation "Included"; if the **-x** keyletter is used, excluded deltas are listed following the notation "Excluded".

TABLE 1. Determination of SCCS Identification String

| SID*<br>Specified | -b Keyletter<br>Used† | Other<br>Conditions                                   | SID<br>Retrieved | SID of Delta<br>to be Created |
|-------------------|-----------------------|-------------------------------------------------------|------------------|-------------------------------|
| none‡             | no                    | R defaults to mR                                      | mR.mL            | mR.(mL+1)                     |
| none‡             | yes                   | R defaults to mR                                      | mR.mL            | mR.mL.(mB+1).1                |
| R                 | no                    | R > mR                                                | mR.mL            | R.1***                        |
| R                 | no                    | R = mR                                                | mR.mL            | mR.(mL+1)                     |
| R                 | yes                   | R > mR                                                | mR.mL            | mR.mL.(mB+1).1                |
| R                 | yes                   | R = mR                                                | mR.mL            | mR.mL.(mB+1).1                |
| R                 | -                     | R < mR and<br>R does <i>not</i> exist<br>Trunk succ.# | hR.mL**          | hR.mL.(mB+1).1                |
| R                 | -                     | in release > R<br>and R exists                        | R.mL             | R.mL.(mB+1).1                 |
| R.L               | no                    | No trunk succ.                                        | R.L              | R.(L+1)                       |
| R.L               | yes                   | No trunk succ.                                        | R.L              | R.L.(mB+1).1                  |
| R.L               | -                     | Trunk succ.<br>in release ≥ R                         | R.L              | R.L.(mB+1).1                  |
| R.L.B             | no                    | No branch succ.                                       | R.L.B.mS         | R.L.B.(mS+1)                  |
| R.L.B             | yes                   | No branch succ.                                       | R.L.B.mS         | R.L.(mB+1).1                  |
| R.L.B.S           | no                    | No branch succ.                                       | R.L.B.S          | R.L.B.(S+1)                   |
| R.L.B.S           | yes                   | No branch succ.                                       | R.L.B.S          | R.L.(mB+1).1                  |
| R.L.B.S           | -                     | Branch succ.                                          | R.L.B.S          | R.L.(mB+1).1                  |

- \* "R", "L", "B", and "S" are the "release", "level", "branch", and "sequence" components of the SID, respectively; "m" means "maximum". Thus, for example, "R.mL" means "the maximum level number within release R"; "R.L.(mB+1).1" means "the first sequence number on the *new* branch (i.e., maximum branch number plus one) of level L within release R". Note that if the SID specified is of the form "R.L", "R.L.B", or "R.L.B.S", each of the specified components *must* exist.
- \*\* "hR" is the highest *existing* release that is lower than the specified, *nonexistent*, release R.
- \*\*\* This is used to force creation of the *first* delta in a *new* release.
- # Successor.
- † The -b keyletter is effective only if the b flag (see *admin* (CP)) is present in the file. An entry of - means "irrelevant".
- ‡ This case applies if the d (default SID) flag is *not* present in the file. If the d flag *is* present in the file, then the SID obtained from the d flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

## Identification Keywords

---

Identifying information is inserted into the text retrieved from the SCCS file by replacing *identification keywords* with their value whenever they occur. The following keywords may be used in the text stored in an SCCS file:

### Keyword Value

|     |                                                                                                                                                       |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| %M% | Module name: either the value of the m flag in the file (see <i>admin</i> (CP)), or if absent, the name of the SCCS file with the leading s. removed. |
| %I% | SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text.                                                                                    |
| %R% | Release.                                                                                                                                              |
| %L% | Level.                                                                                                                                                |
| %B% | Branch.                                                                                                                                               |
| %S% | Sequence.                                                                                                                                             |
| %D% | Current date (YY/MM/DD).                                                                                                                              |
| %H% | Current date (MM/DD/YY).                                                                                                                              |
| %T% | Current time (HH:MM:SS).                                                                                                                              |
| %E% | Date newest applied delta was created (YY/MM/DD).                                                                                                     |
| %G% | Date newest applied delta was created (MM/DD/YY).                                                                                                     |



|     |                                                                                                                                                                                                                                   |
|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| %U% | Time newest applied delta was created (HH:MM:SS).                                                                                                                                                                                 |
| %Y% | Module type: value of the <i>t</i> flag in the SCCS file (see <i>admin</i> (CP)).                                                                                                                                                 |
| %F% | SCCS file name.                                                                                                                                                                                                                   |
| %P% | Fully qualified SCCS file name.                                                                                                                                                                                                   |
| %Q% | The value of the <i>q</i> flag in the file (see <i>admin</i> (CP)).                                                                                                                                                               |
| %C% | Current line number. This keyword is intended for identifying messages output by the program such as "this should not have happened" type errors. It is <i>not</i> intended to be used on every line to provide sequence numbers. |
| %Z% | The 4-character string <i>@(#)</i> recognizable by <i>what</i> (CP).                                                                                                                                                              |
| %W% | A shorthand notation for constructing <i>what</i> (CP) strings for system program files.<br>%W% = %Z%%M%<horizontal-tab>%I%                                                                                                       |
| %A% | Another shorthand notation for constructing <i>what</i> (CP) strings for non-UNIX-type system program files.<br>%A% = %Z%%Y% %M% %I% %Z%                                                                                          |

Several auxiliary files may be created by *get*. These files are known generically as the *g-file*, *l-file*, *p-file*, and *z-file*. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form *s.module-name*; the auxiliary files are named by replacing the leading *s* with the tag. The *g-file* is an exception to this scheme: the *g-file* is named by removing the *s*. prefix. For example, *s.xyz.c*, the auxiliary file names would be *xyz.c*, *l.xyz.c*, *p.xyz.c*, and *z.xyz.c*, respectively.

The *g-file*, which contains the generated text, is created in the current directory (unless the *-p* keyletter is used). A *g-file* is created in all cases, whether or not any lines of text were generated by the *get*. It is owned by the real user. If the *-k* keyletter is used or implied, its mode is 644; otherwise its mode is 444. Only the real user needs write permission in the current directory.

The *l-file* contains a table showing which deltas were applied in generating the retrieved text. The *l-file* is created in the current directory if the *-l* keyletter is used; its mode is 444 and it is owned by the real user. Only the real user needs write permission in the current directory.

Lines in the *l-file* have the following format:

- a. A blank character if the delta was applied;  
\* otherwise.
- b. A blank character if the delta was applied or was not applied and ignored;  
\* if the delta was not applied and was not ignored.

- c. A code indicating a "special" reason why the delta was or was not applied:
  - "I": Included.
  - "X": Excluded.
  - "C": Cut off (by a -c keyletter).
- d. Blank.
- e. SCCS identification (SID).
- f. Tab character.
- g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.
- h. Blank.
- i. Login name of person who created *delta*.

The comments and **MR** data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The *p-file* is used to pass information resulting from a *get* with a -e keyletter along to *delta*. Its contents are also used to prevent a subsequent execution of *get* with a -e keyletter for the same SID until *delta* is executed or the joint edit flag, **j**, (see *admin*(CP)) is set in the SCCS file. The *p-file* is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the *p-file* is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the *get* was executed, followed by a blank and the -i keyletter argument if it was present, followed by a blank and the -x keyletter argument if it was present, followed by a new-line. There can be an arbitrary number of lines in the *p-file* at any time; no two lines can have the same new delta SID.

The *z-file* serves as a *lock-out* mechanism against simultaneous updates. Its contents are the binary (2 bytes) process ID of the command (i.e., *get*) that created it. The *z-file* is created in the directory containing the SCCS file for the duration of *get*. The same protection restrictions as those for the *p-file* apply for the *z-file*. The *z-file* is created mode 444.

## Files

---

|        |                                                                                                        |
|--------|--------------------------------------------------------------------------------------------------------|
| g-file | Existed before the execution of <i>delta</i> ; removed after completion of <i>delta</i> .              |
| p-file | Existed before the execution of <i>delta</i> ; may exist after completion of <i>delta</i> .            |
| q-file | Created during the execution of <i>delta</i> ; removed after completion of <i>delta</i> .              |
| x-file | Created during the execution of <i>delta</i> ; renamed to SCCS file after completion of <i>delta</i> . |



|                |                                                                                               |
|----------------|-----------------------------------------------------------------------------------------------|
| z-file         | Created during the execution of <i>delta</i> ; removed during the execution of <i>delta</i> . |
| d-file         | Created during the execution of <i>delta</i> ; removed after completion of <i>delta</i> .     |
| /usr/bin/bdiff | Program to compute differences between the “gotten” file and the <i>g-file</i> .              |

## See Also

---

admin(CP), delta(CP), prs(CP), what(CP).

## Notes

---

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user does not, then only one file may be named when the *-e* keyletter is used.

## Standards Conformance

---

*get* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## hdr

---

display selected parts of an object file

### Syntax

---

**hdr** [ **-dhmprsSt** ] *file* ...

### Description

---

*hdr* displays object file headers, symbol tables, and text or data relocation records in ASCII formats. It also prints out seek positions for the various segments in the object file. **A.out**, **x.out**, and **x.out** segmented formats and archives are understood.

The symbol table format consists of six fields. In **a.out** formats the third field is missing. The first field is the symbol's index or position in the symbol table, printed in decimal. The index of the first entry is zero. The second field is the type, printed in hexadecimal. The third field is the **s seg** field, printed in hexadecimal. The fourth field is the symbol's value in hexadecimal. The fifth field is a single character that represents the symbol's type as in *nm*(CP), except **C** common is not recognized as a special case of undefined. The last field is the symbol name.

If long form relocation is present, the format consists of six fields. The first is the descriptor, printed in hexadecimal. The second is the symbol ID or index, in decimal. This field is used for external relocations as an index into the symbol table. It should reference an undefined symbol table entry. The third field is the position, or offset, within the current segment at which relocation is to take place; it is printed in hexadecimal. The fourth field is the name of the segment referenced in the relocation: **text**, **data**, **bss**, or **EXT** for external. The fifth field is the size of relocation: **byte**, **word** (2 bytes), or **long**. The last field will indicate, if present, that the relocation is relative.

If short form relocation is present, the format consist of three fields. The first field is the relocation command in hexadecimal. The second field contains the name of the segment referenced: **text** or **data**. The last field indicates the size of relocation: **word** or **long**.

The *hdr* options are:

- d** Prints the data relocation records.
- h** Prints the object file header and extended header. Each field in the header or extended header is labeled. This is the default option.



- m** Prints the text and data segments. This option is similar to the **-S** option but only for text and data segments.
- p** Prints the seek positions as defined by macros in the include file **<a.out.h>**.
- r** Prints both text and data relocation records.
- s** Prints the symbol table.
- S** Prints the file segment table with a header. (Only applicable to **x.out** segmented executable files.)
- t** Prints the text relocation records.

## See Also

---

**a.out(F)**, **nm(CP)**.

## Notes

---

*hdr* is only for use with object files created under UNIX type environments.

## help

---

Asks for help about SCCS commands

### Syntax

---

**help** [args]

### Description

---

*help* finds information to explain a message from an SCCS command or explain the use of a command. Zero or more arguments may be supplied. If no arguments are given, *help* will prompt for one.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names. There are the following types of arguments:

- |        |                                                                                                                                                                                                                            |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| type 1 | Begins with nonnumerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (e.g., <b>ge6</b> , for message 6 from the <i>get</i> command). |
| type 2 | Does not contain numerics (as a command, such as <b>get</b> )                                                                                                                                                              |
| type 3 | Is all numeric (e.g., <b>212</b> )                                                                                                                                                                                         |

The response of the program will be the explanatory information related to the argument, if there is any.

When all else fails, try "help stuck".

### Files

---

|                            |                                            |
|----------------------------|--------------------------------------------|
| <code>/usr/lib/help</code> | Directory containing files of message text |
|----------------------------|--------------------------------------------|



# i286emul

---

emulate 80286

## Syntax

---

**i286emul** [ arg ... ] prog286

## Description

---

*i286emul* is an emulator that allows programs from System V Release 2 or Release 3 on the Intel 80286 to run on System V Release 3 on the Intel 80386.

The system recognizes an attempt to *exec*(S) a 286 program, and automatically *exec*'s the 286 emulator with the 286 program name as an additional argument. It is not necessary to specify the *i286emul* emulator on the command line. The 286 programs can be invoked using the same command format as on the 286 System V.

*i286emul* reads the 286 program's text and data into memory and maps them through the LDT (via *sysi86*(S)) as 286 text and data segments. It also sets callgate 89 in the GDT (which is used by 286 programs for system calls) to point to a routine in *i286emul*. *i286emul* starts the 286 program by jumping to its entry point.

When the 286 program attempts to do a system call, *i286emul* takes control. It does any conversions needed between the 286 system call and the equivalent 386 system call, and performs the 386 system call. The results are converted to the form the 286 program expects, and the 286 program is resumed.

The following are some of the differences between a program running on a 286 and a 286 program using *i286emul* on a 386:

A 286 program under *i286emul* always has 64k in the stack segment if it is a large-model process, or 64k in the data segment if it is a small-model process.

System calls and signal handling use more space on the stack under *i286emul* than it does on a 286.

Attempts to unlink or write on the 286 program will fail on the 286 with ETXTBSY. Under *i286emul*, they will not fail.

*ptrace*(S) is not supported under *i286emul*.

The 286 program must be readable for the emulator to read it.

## Files

---

/bin/i286emul

The emulator must have this name and be in **/bin** if it is to be automatically invoked when *exec*(S) is used on a 286 program.

## Notes

---

The signal mechanism under the emulator is the System V release 2 signal mechanism rather than the System V release 3 mechanism.



# ld

---

invokes the link editor

## Syntax

---

**ld** [options] filename

## Description

---

The *ld* command combines several object files into one, performs relocation, resolves external symbols, and supports symbol table information for symbolic debugging. It creates an executable program by combining one or more object files and copying the executable result to the file **a.out**. The *filename* must name an object or library file. By convention these names have the “.o” (for object) or “.a” (for archive library) extensions. If more than one name is given, the names must be separated by one or more spaces. If any input file, *filename*, is not an object file, *ld* assumes it is either an archive library or a text file containing link editor directives. By default, the file **a.out** is executable if no errors occurred during the load. If errors occur while linking, *ld* displays an error message; the resulting **a.out** file is unexecutable.

*ld* concatenates the contents of the given object files in the order given in the command line. Library files in the command line are examined only if there are unresolved external references encountered from previous object files.

The library is searched iteratively to satisfy as many references as possible and only those routines that define unresolved external references are concatenated. The library (archive) symbol table (see *ar(F)*) is searched sequentially with as many passes as are necessary to resolve external references which can be satisfied by library members. Thus, the ordering of library members is functionally unimportant, unless there exist multiple library members defining the same external symbol. The library may be either a relocatable archive library or a shared library. Object and library files are processed at the point they are encountered in the argument list, so the order of files in the command line is important. In general, all object files should be given before library files. *ld* sets the entry point of the resulting program to the beginning of the first routine.

*ld* should be invoked using the *cc*(CP) command instead of invoking it directly. *cc* invokes *ld* as the last step of compilation, providing all the necessary C-language support routines. Invoking *ld* directly is not recommended since failure to give command line arguments in the correct order can result in errors.

## Generating COFF vs. x.out Binaries

---

When **ld** is called, it scans all the object files that are to be linked. If they are all COFF objects, then the resulting binary will be in COFF format. If any of the object files to be linked are in *x.out* format, any COFF modules in the group will be converted to *x.out* and the resulting binary will be in *x.out* format.

## Common Options

---

The following options are recognized by *ld*, and are common to producing both *COFF* and *x.out* binaries. Refer to the sections "Linking COFF Binaries" and "Linking *x.out* Binaries" for options specific to producing these binaries.

**-o *name***

Sets the executable program filename to *name* instead of **a.out**.

**-r XENIX VERSION:** Invokes the incremental linker, **/lib/ldr**, with the arguments passed to **ld** to produce a relocatable output file.

**AT&T VERSION:** Retains relocation entries in the output object file. Relocation entries must be saved if the output file is to become an input file in a subsequent *ld* run. The link editor will not complain about unresolved references, and the output file will not be executable.

**-s** Strips line number entries and symbol table information from the output object file.

**-u *symbol***

Designates the specified *symbol* as undefined. This is useful for loading entirely from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine. The placement of this option on the *ld* line is significant; it must be placed before the library which will define the symbol.

**-V** Outputs a message giving information about the version of *ld* being used.

**-SE[MENTS]:*number***

Controls the number of segments that the linker allows a program to have. The default is 128, but you can set *number* to any value (decimal, octal, or hexadecimal) in the range 1-1024 (decimal).



For each segment, the linker must allocate some space to keep track of segment information. When you set the segment limit higher than 128, the linker allocates more space for segment information. For programs with fewer than 128 segments, you can keep the storage requirements of the linker at the lowest level possible by setting *number* to reflect the actual number of segments in the program. The linker displays an error message if the number of segments allocated is too high for the amount of memory the linker has available.

**-M[AP] [:*number*]**

Creates a map file. This option is equivalent to using the **-Fm** option with the **cc** command, except that you can give a *number* argument with the **-M** option. The *number* argument is any positive integer (decimal, octal, or hexadecimal) up to 65,535 (decimal) specifying how many symbols are sorted in the map listing. If no *number* argument is given, a maximum of 2048 symbols is sorted. (In practice, the number of sorted symbols is limited by the amount of free heap space.) If a *number* argument is given, the alphabetical list of symbols does not appear in the map file.

**-LI[NENUMBERS]**

Creates a map file and includes the line numbers and associated addresses of the source program. This option is equivalent to using the **-Zd** option with the **cc** command.

**-ST[BACK]:*number***

Specifies the size of the stack for your program, where *number* is any positive value (decimal, octal, or hexadecimal) up to 65,535 (decimal) representing the size, in bytes, of the stack. This option is equivalent to using the **-F** option of the **cc** command.

## Linking COFF Binaries

---

The following options are recognized by *ld* for linking *COFF* binaries:

**-e *epsym***

Set the default entry point address for the output file to be that of the symbol *epsym*.

**-f *fill***

Set the default fill pattern for “holes” within an output section as well as initialized *bss* sections. The argument *fill* is a two-byte constant.

**-l*x*** Search a library **lib*x*.a**, where *x* is up to nine characters. A library is searched when its name is encountered, so the placement of a **-l** is significant. By default, libraries are located in *LIBDIR* or *LLIB-DIR*.



- m Produce a map or listing of the input/output sections on the standard output.
- a Create an absolute file. This is the default if the -r option is not used. Used with the -r option, -a allocates memory for common symbols.
- t Turn off the warning about multiply-defined symbols that are not the same size.
- x Do not preserve local symbols in the output symbol table; enter external and static symbols only. This option saves some space in the output file.
- z Do not bind anything to address zero. This option will allow run-time detection of null pointers.
- L *dir*  
Change the algorithm of searching for *libx.a* to look in *dir* before looking in *LIBDIR* and *LLIBDIR*. This option is effective only if it precedes the -l option on the command line.
- M  
Output a message for each multiply-defined external definition.
- N Put the text section at the beginning of the text segment rather than after all header information, and put the data section immediately following text in the core image.
- VS *num*  
Use *num* as a decimal version stamp identifying the *a.out* file that is produced. The version stamp is stored in the optional header.
- Y[LU],*dir*  
Change the default directory used for finding libraries. If L is specified, the first default directory which *ld* searches, *LIBDIR*, is replaced by *dir*. If U is specified and *ld* has been built with a second default directory, *LLIBDIR*, then that directory is replaced by *dir*. If *ld* was built with only one default directory and U is specified a warning is printed and the option is ignored.

## Linking x.out Binaries

---

The user must make sure that the most recent library versions have been processed with *ranlib*(CP) before linking. Library files for *x.out* format binaries must be in *ranlib*(CP) format, that is, the first member must be named *\_\_SYMDEF*, which is a dictionary for the library. *ld* compares the modification dates of the library and the *\_\_SYMDEF* entry, so if object files have been added to the library since *\_\_SYMDEF* was created, the link may result in an "invalid object module" that cannot run.

The following options are recognized by *ld* for linking *x.out* binaries:

**-A *num***

Creates a standalone program whose expected load address (in hexadecimal) is *num*. This option sets the absolute flag in the header of the *a.out* file. Such program files can only be executed as standalone programs. Options **-A** and **-F** are mutually exclusive.

**-B *num***

Sets the text selector bias to the specified hexadecimal number.

**-c *num***

Alters the default target CPU in the *x.out* header. *num* can be 0, 1, 2, or 3 indicating 8086, 80186, 80286 and 80386 processors, respectively. The default on 8086/80286 systems is 0. The default on 80386 systems is 3. Note that this option only alters the default; if object modules containing code for a higher numbered processor are linked, then that will take precedence over the default.

**-C**

Causes the link editor to ignore the case of symbols.

**-D *num***

Sets the data selector bias to the specified hexadecimal number.

**-F *num***

Sets the size of the program stack to *num* bytes where *num* is a hexadecimal number. This option is ignored for 80386 programs which have a variable sized stack. By default 8086 programs have a variable stack located at the top of the first data segment, and 80286 programs have a fixed size 4096 byte stack. The **-F** option is incompatible with the **-A** option that cannot be opened by more than one user at the same time.

**-g** Includes symbolic information for *sdb*.

**-i** Creates separate instruction and data spaces for small model programs. When the output file is executed, the program text and data areas are allocated separate physical segments. The text portion will be read-only and shared by all users executing the file.

**-La**

Sets advisory file locking. Advisory locking is used on files with access modes that do not require mandatory locking.

**-Lm**

Sets mandatory file locking. Mandatory file locking is used on files that cannot be opened by more than one process at a time.

**-m *name***

Creates a link map file named *name* that includes public symbols.



**-Mx**

Specifies the memory model. *x* can have the following values:

|   |        |
|---|--------|
| s | small  |
| m | middle |
| l | large  |
| h | huge   |
| e | mixed  |

**-n num**

Truncates symbols to the length specified by *num*.

**-N num**

Sets the pagesize to hex-*num* (which should be a multiple of 512) - the default is 1024 for 80386 programs. 8086/80186/80286 programs do not normally have page-aligned *x.out* files and the default for these is 0.

**-P**

Disables packing of segments

**-R** Ensures that the relocation table is of non-zero size. Important for 8086 compatibility.

**-Rd num**

Specify the data segment relocation offset (80386 only). *num* is hexadecimal.

**-Rt num**

Specify the text segment relocation offset (80386 only) *num* is hexadecimal.

**-S num**

Sets the maximum number of segments to *num*. If no argument is given, the default is 128.

## Files

---

/bin/ld

*LIBDIR*/libx.a

libraries

*LLIBDIR*/libx.a

libraries

a.out

output file

*LIBDIR*

usually /lib

*LLIBDIR*

usually /usr/lib

## See Also

---

as(CP), cc(CP), masm(CP), mkshlib(CP), ranlib(CP), exit(S), end(S), a.out(F), ar(F).

## Notes

---

Through its options and input directives, the common link editor gives users great flexibility; however, those who use the input directives must assume some added responsibilities. Input directives and options should insure the following properties for programs:

- C defines a zero pointer as null. A pointer to which zero has been assigned must not point to any object. To satisfy this, users must not place any object at virtual address zero in the program's address space.
- When the link editor is called through *cc*(CP), a startup routine is linked with the user's program. This routine calls *exit*( ) [see *exit*(S)] after execution of the main program. If the user calls the link editor directly, then the user must insure that the program always calls *exit*( ) rather than falling through the end of the entry routine.

The symbols *etext*, *edata*, and *end* (see *end*(S)) are reserved and are defined by the link editor. It is incorrect for a user program to redefine them.

If the link editor does not recognize an input file as an object file or an archive file, it will assume that it contains link editor directives and will attempt to parse it. This will occasionally produce an error message complaining about "syntax errors".

Arithmetic expressions may only have one forward referenced symbol per expression.

If you are using XENIX binaries, please refer to the manual entry for this utility in the *XENIX Development Guide* for information on the appropriate usage with XENIX binaries.

## Standards Conformance

---

*ld* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



## lex

---

generate programs for simple lexical tasks

### Syntax

---

`lex [ -rctvn ] [ file ] ...`

### Description

---

The *lex* command generates programs to be used in simple lexical analysis of text.

The input *files* (standard input default) contain strings and expressions to be searched for, and C text to be executed when strings are found.

A file **lex.yy.c** is generated which, when loaded with the library, copies the input to the output except when a string specified in the file is found; then the corresponding program text is executed. The actual string matched is left in *yytext*, an external character array. Matching is done in order of the strings in the file. The strings may contain square brackets to indicate character classes, as in **[abx-z]** to indicate **a**, **b**, **x**, **y**, and **z**; and the operators **\***, **+**, and **?** mean respectively any non-negative number of, any positive number of, and either zero or one occurrence of, the previous character or character class. The character **.** is the class of all ASCII characters except new-line. Parentheses for grouping and vertical bar for alternation are also supported. The notation **r{d,e}** in a rule indicates between **d** and **e** instances of regular expression **r**. It has higher precedence than **/**, but lower than **\***, **?**, **+**, and concatenation. Thus **[a-zA-Z]+** matches a string of letters. The character **^** at the beginning of an expression permits a successful match only immediately after a new-line, and the character **\$** at the end of an expression requires a trailing new-line. The character **/** in an expression indicates trailing context; only the part of the expression up to the slash is returned in *yytext*, but the remainder of the expression must follow in the input stream. An operator character may be used as an ordinary symbol if it is within **"** symbols or preceded by **\**.

Three subroutines defined as macros are expected: **input()** to read a character; **unput(c)** to replace a character read; and **output(c)** to place an output character. They are defined in terms of the standard streams, but you can override them. The program generated is named **yylex()**, and the library contains a **main()** which calls it. The action **REJECT** on the right side of the rule causes this match to be rejected and the next suitable match executed; the function **yymore()** accumulates additional characters into the same *yytext*; and the function **yyless(p)** pushes back the portion of the string matched beginning at

*p*, which should be between *yytext* and *yytext+yy leng*. The macros *input* and *output* use files *yyin* and *yyout* to read from and write to, defaulted to *stdin* and *stdout*, respectively.

Any line beginning with a blank is assumed to contain only C text and is copied; if it precedes `%%`, it is copied into the external definition area of the `lex.yy.c` file. All rules should follow a `%%`, as in YACC. Lines preceding `%%` which begin with a non-blank character define the string on the left to be the remainder of the line; it can be called out later by surrounding it with `{}`. Note that curly brackets do not imply parentheses; only string substitution is done.

## Example

---

```
D [0-9]
%%
if printf("IF statement\n");
[a-z]+ printf("tag, value %s\n",yytext);
0{D}+ printf("octal number %s\n",yytext);
{D}+ printf("decimal number %s\n",yytext);
"++" printf("unary op\n");
"+" printf("binary op\n");
"/*" skipcommnts();
%%
skipcommnts()
{
 for (;;)
 {
 while (input() != '*')
 ;
 if (input() != '/')
 unput(yytext[yytext-1]);
 else
 return;
 }
}
```

The external names generated by *lex* all begin with the prefix *yy* or *YY*.

The flags must appear before any files. The flag **-r** indicates RATFOR actions, **-c** indicates C actions and is the default, **-t** causes the `lex.yy.c` program to be written instead to standard output, **-v** provides a one-line summary of statistics, **-n** will not print out the **-v** summary. Multiple files are treated as a single file. If no files are specified, standard input is used.



Certain table sizes for the resulting finite state machine can be set in the definitions section:

**%p** *n* number of positions is *n* (default 2500)

**%n** *n* number of states is *n* (500)

**%e** *n* number of parse tree nodes is *n* (1000)

**%a** *n* number of transitions is *n* (2000)

**%k** *n* number of packed character classes is *n* (1000)

**%o** *n* size of output array is *n* (3000)

The use of one or more of the above automatically implies the **-v** option, unless the **-n** option is used.

## See Also

---

yacc(CP).

## Notes

---

The **-r** option is not yet fully operational.

## Standards Conformance

---

*lex* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

# lint

a C program checker

## Syntax

`lint [ option ] ... file ...`

## Description

The *lint* command attempts to detect features of the C program files that are likely to be bugs, non-portable, or wasteful. It also checks type usage more strictly than the compilers. Among the things that are currently detected are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions that return values in some places and not in others, functions called with varying numbers or types of arguments, and functions whose values are not used or whose values are used but none returned.

Arguments whose names end with `.c` are taken to be C source files. Arguments whose names end with `.ln` are taken to be the result of an earlier invocation of *lint* with either the `-c` or the `-o` option used. The `.ln` files are analogous to `.o` (object) files that are produced by the *cc*(CP) command when given a `.c` file as input. Files with other suffixes are warned about and ignored.

The *lint* command will take all the `.c`, `.ln`, and `llib-lx.ln` (specified by `-lx`) files and process them in their command line order. By default, *lint* appends the standard C lint library (`llib-lc.ln`) to the end of the list of files. However, if the `-p` option is used, the portable C lint library (`llib-port.ln`) is appended instead. When the `-c` option is not used, the second pass of *lint* checks this list of files for mutual compatibility. When the `-c` option is used, the `.ln` and the `llib-lx.ln` files are ignored.

Any number of *lint* options may be used, in any order, intermixed with file-name arguments. The following options are used to suppress certain kinds of complaints:

- a Suppress complaints about assignments of long values to variables that are not long.
- b Suppress complaints about **break** statements that cannot be reached. (Programs produced by *lex* or *yacc* will often result in many such complaints.)



- h Do not apply heuristic tests that attempt to intuit bugs, improve style, and reduce waste.
- u Suppress complaints about functions and external variables used and not defined, or defined and not used. (This option is suitable for running *lint* on a subset of files of a larger program.)
- v Suppress complaints about unused arguments in functions.
- x Do not report variables referred to by external declarations but never used.

The following arguments alter *lint*'s behavior:

.rm]B

Include additional lint library **llib-lx.ln**. For example, you can include a lint version of the math library **llib-lm.ln** by inserting **-lm** on the command line. This argument does not suppress the default use of **llib-lc.ln**. These lint libraries must be in the assumed directory. This option can be used to reference local lint libraries and is useful in the development of multifile projects.

.rm]B

Do not check compatibility against either the standard or the portable lint library.

.rm]B

Attempt to check portability to other dialects (IBM and GCOS) of C. Along with stricter checking, this option causes all non-external names to be truncated to eight characters and all external names to be truncated to six characters and one case.

.rm]B

Cause *lint* to produce a **.ln** file for every **.c** file on the command line. These **.ln** files are the product of *lint*'s first pass only, and are not checked for inter-function compatibility.

.rm]B

Cause *lint* to create a lint library with the name **llib-llib.ln**. The **-c** option nullifies any use of the **-o** option. The lint library produced is the input that is given to *lint*'s second pass. The **-o** option simply causes this file to be saved in the named lint library. To produce a **llib-llib.ln** without extraneous messages, use of the **-x** option is suggested. The **-v** option is useful if the source file(s) for the lint library are just external interfaces (for example, the way the file **llib-lc** is written). These option settings are also available through the use of "lint comments" (see below).

The **-D**, **-U**, and **-I** options of *cpcp*(CP) and the **-g** and **-O** options of *cc*(CP) are also recognized as separate arguments. The **-g** and **-O** options are ignored, but, by recognizing these options, *lint*'s behavior is closer to that of the command. Other options are warned about and

ignored. The preprocessor symbol “*lint*” is defined to allow certain questionable code to be altered or removed for *lint*. Therefore, the symbol “*lint*” should be thought of as a reserved word for all code that is planned to be checked by *lint*.

Certain conventional comments in the C source will change the behavior of *lint*:

```
/*NOTREACHED*/
 at appropriate points stops comments about unreachable code.
 [This comment is typically placed just after calls to functions
 like exit(S).]
```

```
/*VARARGSn*/
 suppresses the usual checking for variable numbers of argu-
 ments in the following function declaration. The data types of
 the first n arguments are checked; a missing n is taken to be 0.
```

```
/*ARGSUSED*/
 turns on the -v option for the next function.
```

```
/*LINTLIBRARY*/
 at the beginning of a file shuts off complaints about unused
 functions and function arguments in this file. This is equivalent
 to using the -v and -x options.
```

The *lint* command produces its first output on a per-source-file basis. Complaints regarding included files are collected and printed after all source files have been processed. Finally, if the -c option is not used, information gathered from all input files is collected and checked for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its included files, the source file name will be printed followed by a question mark.

The behavior of the -c and the -o options allows for incremental use of *lint* on a set of C source files. Generally, one invokes *lint* once for each source file with the -c option. Each of these invocations produces a .ln file for each .c file, and prints all messages that are about just that source file. After all the source files have been separately run through *lint*, it is invoked once more (without the -c option), listing all the .ln files with the needed -lx options. This will print all the interfile inconsistencies. This scheme works well with *make*(CP); it allows *make* to be used to *lint* only the source files that have been modified since the last time the set of source files were *linted*.

## Files

---

|                            |                                                                                                                                             |
|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <i>LLIBDIR</i>             | the directory where the lint libraries specified by the <i>-lx</i> option must exist, usually <i>/usr/lib</i>                               |
| <i>LLIBDIR/lint[12]</i>    | first and second passes                                                                                                                     |
| <i>LLIBDIR/lib-lc.ln</i>   | declarations for C Library functions (binary format; source is in <i>LLIBDIR/lib-lc</i> )                                                   |
| <i>LLIBDIR/lib-port.ln</i> | declarations for portable functions (binary format; source is in <i>LLIBDIR/lib-port</i> )                                                  |
| <i>LLIBDIR/lib-lm.ln</i>   | declarations for Math Library functions (binary format; source is in <i>LLIBDIR/lib-lm</i> )                                                |
| <i>TMPDIR/*lint*</i>       | temporaries                                                                                                                                 |
| <i>TMPDIR</i>              | usually <i>/usr/tmp</i> but can be redefined by setting the environment variable <b>TMPDIR</b> [see <i>tempnam()</i> in <i>tmpnam(S)</i> ]. |

## See Also

---

*cc*(CP), *cpp*(CP), *make*(CP)

## Notes

---

*exit(S)*, *setjmp(S)*, and other functions that do not return are not understood; this causes various lies.

## Standards Conformance

---

*lint* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



## list

---

produce C source listing from a common object file

### Syntax

---

**list** [-V] [-h] [-F function] source-file . . . [object-file]

### Description

---

The *list* command produces a C source listing with line number information attached. If multiple C source files were used to create the object file, *list* will accept multiple file names. The object file is taken to be the last non-C source file argument. If no object file is specified, the default object file, **a.out**, will be used.

Line numbers will be printed for each line marked as breakpoint inserted by the compiler (generally, each executable C statement that begins a new line of source). Line numbering begins anew for each function. Line number 1 is always the line containing the left curly brace ( { ) that begins the function body. Line numbers will also be supplied for inner block redeclarations of local variables so that they can be distinguished by the symbolic debugger.

The following options are interpreted by *list* and may be given in any order:

- |                   |                                                                                                         |
|-------------------|---------------------------------------------------------------------------------------------------------|
| <b>-V</b>         | Print, on standard error, the version number of the <i>list</i> command executing.                      |
| <b>-h</b>         | Suppress heading output.                                                                                |
| <b>-Ffunction</b> | List only the named function. The <b>-F</b> option may be specified multiple times on the command line. |

### See Also

---

as(CP), cc(CP), ld(CP).

### Diagnostics

---

The *list* command will produce the error message “list: name: cannot open” if *name* cannot be read. If the source file names do not end in .c, the message is “list: name: invalid C source name”. An invalid object file will cause the message “list: name: bad magic” to be produced. If some or all of the symbolic debugging information is missing, one of the following messages will be printed: “list: name:

symbols have been stripped, cannot proceed", "list: name: cannot read line numbers", and "list: name: not in symbol table". The following messages are produced when *list* has become confused by *#ifdef*'s in the source file: "list: name: cannot find function in symbol table", "list: name: out of sync: too many }", and "list: name: unexpected end-of-file". The error message "list: name: missing or inappropriate line numbers" means that either symbol debugging information is missing, or *list* has been confused by C preprocessor statements.

## Notes

---

Note that this utility operates only on COFF executables. Object files given to *list* must have been compiled with the *-g* option of *cc* (CP).

Since *list* does not use the C preprocessor, it may be unable to recognize function definitions whose syntax has been distorted by the use of C preprocessor macro substitutions.

## lorder

---

find ordering relation for an object library

### Syntax

---

**lorder** file ...

### Description

---

The input is one or more object or library archive *files* (see *ar*(CP)). The standard output is a list of pairs of object file or archive member names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by *tsort*(CP) to find an ordering of a library suitable for one-pass access by *ld*(CP). Note that the link editor *ld*(CP) is capable of multiple passes over an archive in the portable archive format (see *ar*(F)) and does not require that *lorder*(CP) be used when building an archive. The usage of the *lorder*(CP) command may, however, allow for a slightly more efficient access of the archive during the link edit process.

The following example builds a new library from existing *.o* files.

```
ar -cr library `lorder *.o | tsort`
```

### Files

---

*TMPDIR*/\*symref      temporary files

*TMPDIR*/\*symdef      temporary files

*TMPDIR* is usually */usr/tmp* but can be redefined by setting the environment variable **TMPDIR** [see *tempnam*() in *tmpnam*(S)].

### See Also

---

*ar*(CP), *ld*(CP), *tsort*(CP), *ar*(F)

### Note

---

The *lorder* command will accept as input any object or archive file, regardless of its suffix, provided there is more than one input file. If there is but a single input file, its suffix must be *.o*.



## **Standards Conformance**

---

*lorder* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## **lprof**

display line-by-line execution count profile data

### **Syntax**

```
lprof [-p] [-s] [-x] [[-I incdir]] [[-r srcfile]] [-c cntfile]
lprof -m file1.cnt file2.cnt [[filen.cnt]] [-T] -d destfile.cnt
```

### **Description**

*lprof* is a tool for dynamic analysis; that is, the analysis of a program at run time. Specifically, *lprof* identifies the most frequently executed parts of source code and parts of code that are never executed.

*lprof* interprets a profile file (**prog.cnt** by default) produced by the profiled program *prog* (*a.out* by default) that has been compiled with the **-ql** option of *cc*(CP). This *cc* command option arranges for code to be inserted to record run-time behavior and for data to be written to a file at the end of execution.

By default, *lprof* prints a listing of source files (the names of which are stored in the symbol table of the executable file), each line preceded by its line number (in the file) and the number of times it was executed.

The following options may appear singly or be combined in any order:

- p** Print listing, each line preceded by the line number and the number of times it was executed (default). This option can be used together with the **-s** option to print both the source listing and summary information.
- s** Print summary information of percentage of lines of code executed per function.
- x** Instead of printing the execution count numbers for each line, print each line preceded by its line number and a [U] if the line was not executed. If the line was executed, print only the line number.

**-I *incdir***

Look for source or header files in the directory **incdir** in addition to the current directory and the standard place for **#include** files (usually **/usr/include**). You can specify more than one directory with **-I** on one command line.

**-r *srcfile***

Instead of printing all source files, print only those files named in **-r** options (to be used with the **-p** option only). You can specify multiple files with **-r** on one command line.

**-c *cntfile***

Use the file *cntfile* instead of **prog.cnt** as the input profile file.

**-o *prog***

Use the name of the program *prog* instead of the name used when creating the profile file. Because the program name stored in the profile file contains the relative path, this option is necessary if the executable file or profile file has been moved.

### Merging Data Files

*lprof* can also be used to merge data files. The **-m** option must be accompanied with the **-d** option:

**-m *file1.cnt file2.cnt [filen.cnt]* -d *destfile.cnt***

Merge the data files **file1.cnt** through **filen.cnt** by summing the execution counts per line, so that data from several runs can be accumulated. The result is written to **destfile.cnt**. The data files must contain profiling data for the same **prog** (see the **-T** option below).

**-T** Time stamp override. Normally, the time stamps of the executable files being profiled are checked, and data files will not be merged if the time stamps do not match. If **-T** is specified, this check is skipped.

### Controlling the Run Time Profiling Environment

The environment variable **PROFOPTS** provides run time control over profiling. When a profiled program is about to terminate, it examines the value of **PROFOPTS** to determine how the profiling data is to be handled.

The environment variable **PROFOPTS** is a comma-separated list of options interpreted by the program being profiled. If **PROFOPTS** is not defined in the environment, then the default action is taken: the profiling data is saved in a file (with the default name, **prog.cnt**) in the current directory. If **PROFOPTS** is set to the null string, no profiling data is saved. The following are the available options:

**msg=[*y*|*n*]**

If **msg=y** is specified, a message stating that profile data is being saved is printed to *stderr*. If **msg=n** is specified, print only profiling error messages. The default is **msg=y**.



**merge=[y | n]**

If **merge=n** is specified, do not merge data files after successive runs. The data file is overwritten after each execution. If **merge=y** is specified, the data will be merged. The merge will fail if the program has been recompiled; the data file will be left in **TMPDIR**. The default is **merge=n**.

**pid=[y | n]**

If **pid=y** is specified, the name of the data file will include the process ID of the profiled program. This allows the creation of different data files for programs calling *fork(S)*. If **pid=n** is specified, the default name is used. The default is **pid=n**.

**dir=dirname**

Place the data file in the directory **dirname** if this option is specified. Otherwise, the data file is created in the directory that is current at the end of execution.

**file=filename**

Use **filename** as the name of the data file in **dir** created by the profiled program if this option is specified. Otherwise, the default name is used.

## Files

---

|                 |                  |
|-----------------|------------------|
| <b>prog.cnt</b> | for profile data |
| <b>TMPDIR/*</b> | temporary files  |

**TMPDIR** is usually **/usr/tmp**, but can be redefined by setting the environment variable **TMPDIR** [see *tmpnam()* in *tmpnam(S)*].

## See Also

---

**cc(CP)**, **prof(CP)**, **fork(S)**, **tmpnam(S)**

## Warnings

---

For the **-m** option, if **destfile.cnt** exists, its previous contents are destroyed.

Optimizing functions may result in the loss of some line number information and may result in code motions, both of which may make *lprof* information unreliable.

Different parts of one line of a source file may be executed different numbers of times (e.g., the **for** loop below); the count corresponds to the first part of the line. For example, in the following **for** loop:

```

1 [8] for (j = 0; j < 5; j++)
5 [9] sub(j);

```

line 8 consists of three parts. The line count listed, however, is for the initialization part, that is,  $j = 0$ .

*lprof* incorrectly handles the statement immediately following a **for** loop containing a single **if** statement. In the following example, line 8 is executed only once.

```

1 [5] for (i = 0; i < 3; i++)
3 [6] if (i > 3)
0 [7] x = i;
3 [8] i = 0;

```

This problem can be solved by adding curly braces, as follows:

```

1 [5] for (i = 0; i < 3; i++) {
3 [6] if (i > 3)
0 [7] x = i;
3 [8] }
1 [9] i = 0;

```

*lprof* then handles the statement following the **for** loop correctly.

*lprof* does not provide execution information about **case** statements containing only a **break** statement, or about **return** statements without a value.

```

1 [4] switch (i) {
 case 0:
 break;
 default:
0 [8] i = 0;
 }

1 [11] if (i != 0)
 return;

```

# m4

## macroprocessor

---

### Syntax

---

**m4** [ options ] [ files ]

### Description

---

The *m4* command is a macroprocessor intended as a front end for Rat-for, C, and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is -, the standard input is read. The processed text is written on the standard output.

The options and their effects are as follows:

- e Operate interactively. Interrupts are ignored and the output is unbuffered.
- s Enable line sync output for the C preprocessor (#line ...)
- Bint Change the size of the push-back and argument collection buffers from the default of 4,096.
- Hint Change the size of the symbol table hash array from the default of 199. The size should be prime.
- Sint Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.
- Tint Change the size of the token buffer from the default of 512 bytes.

To be effective, these flags must appear before any file names and before any -D or -U flags:

- Dname[=val]  
Defines *name* to *val* or to null in *val*'s absence.
- Uname  
Undefines *name*.

Macro calls have the form:

name(arg1,arg2, ..., argn)



The open parentheses, (, must immediately follow the name of the macro. If the name of a defined macro is not followed by a close parentheses, ), it is deemed to be a call of that macro with no arguments. Potential macro names consist of alphabetic letters, digits, and underscore \_, where the first character is not a digit.

Leading unquoted blanks, tabs, and new-lines are ignored while collecting arguments. Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are taken to be null. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

The *m4* command makes available the following built-in macros. They may be redefined, but once this is done, the original meaning is lost. Their values are null unless otherwise stated.

|          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| define   | the second argument is installed as the value of the macro whose name is the first argument. Each occurrence of \$ <i>n</i> in the replacement text, where <i>n</i> is a digit, is replaced by the <i>n</i> -th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string; \$# is replaced by the number of arguments; \$* is replaced by a list of all the arguments separated by commas; \$@ is like \$*, but each argument is quoted (with the current quotes). |
| undefine | removes the definition of the macro named in its argument.                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| defn     | returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.                                                                                                                                                                                                                                                                                                                                                                                                 |
| pushdef  | like <i>define</i> , but saves any previous definition.                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| popdef   | removes current definition of its argument(s), exposing the previous one, if any.                                                                                                                                                                                                                                                                                                                                                                                                                         |
| ifdef    | if the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word <i>unix</i> is predefined on some versions of <i>m4</i> .                                                                                                                                                                                                                                                                                             |

|             |                                                                                                                                                                                                                                                                                                                                           |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| shift       | returns all but its first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.                                                                                                                                        |
| changequote | change quote symbols to the first and second arguments. The symbols may be up to five characters long. <i>changequote</i> without arguments restores the original values (that is, ` `).                                                                                                                                                  |
| changecom   | change left and right comment markers from the default # and new-line. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes new-line. With two arguments, both markers are affected. Comment markers may be up to five characters long. |
| divert      | <i>m4</i> maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The <i>divert</i> macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.        |
| undivert    | causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.                                                                                                                                              |
| divnum      | returns the value of the current output stream.                                                                                                                                                                                                                                                                                           |
| dnl         | reads and discards characters up to and including the next new-line.                                                                                                                                                                                                                                                                      |
| ifelse      | has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6, and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.                    |
| incr        | returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.                                                                                                                                                                                  |
| decr        | returns the value of its argument decremented by 1.                                                                                                                                                                                                                                                                                       |
| eval        | evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, -, *, /, %, ^ (exponentiation), bitwise &,  , ^, and ~; relational; parentheses. Octal and hex numbers may be                                                                                                                           |



specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.

|          |                                                                                                                                                                                                                                                                                |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| len      | returns the number of characters in its argument.                                                                                                                                                                                                                              |
| index    | returns the position in its first argument where the second argument begins (zero origin), or -1 if the second argument does not occur.                                                                                                                                        |
| substr   | returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string. |
| translit | transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.                                                                                                                   |
| include  | returns the contents of the file named in the argument.                                                                                                                                                                                                                        |
| sinclude | is identical to <i>include</i> , except that it says nothing if the file is inaccessible.                                                                                                                                                                                      |
| syscmd   | executes the system command given in the first argument. No value is returned.                                                                                                                                                                                                 |
| sysval   | is the return code from the last call to <i>syscmd</i> .                                                                                                                                                                                                                       |
| maketemp | fills in a string of XXXXX in its argument with the current process ID.                                                                                                                                                                                                        |
| m4exit   | causes immediate exit from <i>m4</i> . Argument 1, if given, is the exit code; the default is 0.                                                                                                                                                                               |
| m4wrap   | argument 1 will be pushed back at final EOF; example: <i>m4wrap</i> (`cleanup`)                                                                                                                                                                                                |
| errprint | prints its argument on the diagnostic output file.                                                                                                                                                                                                                             |
| dumpdef  | prints current names and definitions for the named items or for all if no arguments are given.                                                                                                                                                                                 |
| traceon  | with no arguments, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.                                                                                                                                                        |
| traceoff | turns off trace globally and for any macros specified. Macros specifically traced by <i>traceon</i> can be untraced only by specific calls to <i>traceoff</i> .                                                                                                                |



## **See Also**

---

cc(CP), cpp(CP).

## **Standards Conformance**

---

*m4* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

# make

maintain, update, and regenerate groups of programs

## Syntax

**make** [-f *makefile*] [-piksrbetq] [names]

## Description

*make* allows the programmer to maintain, update, and regenerate groups of computer programs. The following is a brief description of all options and some special names:

- f *makefile*** Description file name. *makefile* is assumed to be the name of a description file.
- p** Print out the complete set of macro definitions and target descriptions.
- i** Ignore error codes returned by invoked commands. This mode is entered if the fake target name **.IGNORE** appears in the description file.
- k** Abandon work on the current entry if it fails, but continue on other branches that do not depend on that entry.
- s** Silent mode. Do not print command lines before executing. This mode is also entered if the fake target name **.SILENT** appears in the description file.
- r** Do not use the built-in rules.
- n** No execute mode. Print commands, but do not execute them. Even lines beginning with an **@** are printed.
- b** Compatibility mode for old makefiles.
- e** Environment variables override assignments within makefiles.
- t** Touch the target files (causing them to be up-to-date) rather than issue the usual commands.
- q** Question. The *make* command returns a zero or non-zero status code depending on whether the target file is or is not up-to-date.

**.DEFAULT** If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name **.DEFAULT** are used if it exists.

**.PRECIOUS**

Dependents of this target will not be removed when quit or interrupt are hit.

**.SILENT** Same effect as the **-s** option.

**.IGNORE** Same effect as the **-i** option.

*make* executes commands in *makefile* to update one or more target names. *Name* is typically a program. If no **-f** option is present, **makefile**, **Makefile**, and the Source Code Control System (SCCS) files **s.makefile**, and **s.Makefile** are tried in order. If *makefile* is **-**, the standard input is taken. More than one **-f makefile** argument pair may appear.

*make* updates a target only if its dependents are newer than the target. All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out-of-date.

*makefile* contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, then a **:**, then a (possibly null) list of prerequisite files or dependencies. Text following a **;** and all following lines that begin with a tab are shell commands to be executed to update the target. The first non-empty line that does not begin with a tab or **#** begins a new dependency or macro definition. Shell commands may be continued across lines with the **<backslash><new-line>** sequence. Everything printed by *make* (except the initial tab) is passed directly to the shell as is. Thus,

```
echo a\
b
```

will produce

```
ab
```

exactly the same as the shell would.

Sharp (**#**) and new-line surround comments.



The following *makefile* says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) and a common file **incl.h**:

```
pgm: a.o b.o
 cc a.o b.o -o pgm
a.o: incl.h a.c
 cc -c a.c
b.o: incl.h b.c
 cc -c b.c
```

Command lines are executed one at a time, each by its own shell. The **SHELL** environment variable can be used to specify which shell *make* should use to execute commands. The default is */bin/sh*. The first one or two characters in a command can be the following: **-**, **@**, **-@**, or **@-**. If **@** is present, printing of the command is suppressed. If **-** is present, *make* ignores an error. A line is printed when it is executed unless the **-s** option is present, or the entry **.SILENT:** is in *makefile*, or unless the initial character sequence contains a **@**. The **-n** option specifies printing without execution; however, if the command line has the string **\$(MAKE)** in it, the line is always executed (see discussion of the **MAKEFLAGS** macro under *Environment*). The **-t** (touch) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate *make*. If the **-i** option is present, or the entry **.IGNORE:** appears in *makefile*, or the initial character sequence of the command contains **-**, the error is ignored. If the **-k** option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

The **-b** option allows old makefiles (those written for the old version of *make*) to run without errors.

Interrupt and quit cause the target to be deleted unless the target is a dependent of the special name **.PRECIOUS**.

## Environment

The environment is read by *make*. All variables are assumed to be macro definitions and processed as such. The environment variables are processed before any *makefile* and after the internal rules; thus, macro assignments in a *makefile* override environment variables. The **-e** option causes the environment to override the macro assignments in a *makefile*. Suffixes and their associated rules in the *makefile* will override any identical suffixes in the built-in rules.

The **MAKEFLAGS** environment variable is processed by *make* as containing any legal input option (except **-f** and **-p**) defined for the command line. Further, upon invocation, *make* "invents" the variable if it is not in the environment, puts the current options into it, and passes it

on to invocations of commands. Thus, **MAKEFLAGS** always contains the current input options. This proves very useful for “super-makes”. In fact, as noted above, when the **-n** option is used, the command **\$(MAKE)** is executed anyway; hence, one can perform a **make -n** recursively on a whole software system to see what would have been executed. This is because the **-n** is put in **MAKEFLAGS** and passed to further invocations of **\$(MAKE)**. This is one way of debugging all of the makefiles for a software project without actually doing anything.

### Include Files

If the string *include* appears as the first seven letters of a line in a *makefile*, and is followed by a blank or a tab, the rest of the line is assumed to be a filename and will be read by the current invocation, after substituting for any macros.

### Macros

Entries of the form *string1* = *string2* are macro definitions. *string2* is defined as all characters up to a comment character or an unescaped new-line. Subsequent appearances of **\$(string1[:subst1=[subst2]])** are replaced by *string2*. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional **:subst1=subst2** is a substitute sequence. If it is specified, all non-overlapping occurrences of *subst1* in the named macro are replaced by *subst2*. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines. An example of the use of the substitute sequence is shown under *Libraries*.

### Internal Macros

There are five internally maintained macros that are useful for writing rules for building targets.

- \$\*** The macro **\$\*** stands for the filename part of the current dependent with the suffix deleted. It is evaluated only for inference rules.
- \$@** The **\$@** macro stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.
- \$<** The **\$<** macro is only evaluated for inference rules or the **.DEFAULT** rule. It is the module that is out-of-date with respect to the target (i.e., the “manufactured” dependent file name). Thus, in the **.c.o** rule, the **\$<** macro would evaluate to the **.c** file. An example for making optimized **.o** files from **.c** files is:

```
.c.o:
cc -c -O $*.c
```



or:

```
.c.o:
cc -c -O $<
```

- \$?** The **\$?** macro is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are out-of-date with respect to the target; essentially, those modules which must be rebuilt.
- \$%** The **\$%** macro is only evaluated when the target is an archive library member of the form **lib(file.o)**. In this case, **\$@** evaluates to **lib** and **\$%** evaluates to the library member, **file.o**.

Four of the five macros can have alternative forms. When an upper-case **D** or **F** is appended to any of the four macros, the meaning is changed to "directory part" for **D** and "file part" for **F**. Thus, **\$(@D)** refers to the directory part of the string **\$@**. If there is no directory part, **/** is generated. The only macro excluded from this alternative form is **\$?**.

### Suffixes

Certain names (for example, those ending with **.o**) have inferable prerequisites such as **.c**, **.s**, etc. If no update commands for such a file appear in *makefile*, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, *make* has inference rules which allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. The current default inference rules are:

```
.c .c~ .f .f~ .sh .sh~
.c.o .c.a .c~.o .c~.c .c~.a
.f.o .f.a .f~.o .f~.f .f~.a
.h~.h .s.o .s~.o .s~.s .s~.a .sh~.sh
.l.o .l.c .l~.o .l~.l .l~.c
.y.o .y.c .y~.o .y~.y .y~.c
```

The internal rules for *make* are contained in the source file **rules.c** for the *make* program. These rules can be locally modified. To print out the rules compiled into the *make* on any machine in a form suitable for recompilation, the following command is used:

```
make -fp - 2>/dev/null </dev/null
```

A tilde in the above rules refers to an SCCS file (see *sccsfile(F)*). Thus, the rule **.c~.o** would transform an SCCS C source file into an object file (**.o**). Because the **s.** of the SCCS files is a prefix, it is incompatible with *make*'s suffix point of view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.



A rule with only one suffix (i.e., **.c:**) is the definition of how to build *x* from *x.c*. In effect, the other suffix is null. This is useful for building targets from only one source file (e.g., shell procedures, simple C programs).

Additional suffixes are given as the dependency list for **.SUFFIXES**. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite. The default list is:

**.SUFFIXES:** .o .c .c~ .y .y~ .l .l~ .s .s~ .sh .sh~ .h .h~ .f .f~

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; **.SUFFIXES:** with no dependencies clears the list of suffixes.

### Inference Rules

The first example can be done more briefly.

```
pgm: a.o b.o
 cc a.o b.o -o pgm
a.o b.o: incl.h
```

This is because *make* has a set of internal rules for building files. The user may add rules to this list by simply putting them in the *makefile*.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, **CFLAGS**, **LFLAGS**, and **YFLAGS** are used for compiler options to *cc*(CP), *lex*(CP), and *yacc*(CP), respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix **.o** from a file with suffix **.c** is specified as an entry with **.c.o:** as the target and no dependents. Shell commands associated with the target define the rule for making a **.o** file from a **.c** file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

### Libraries

If a target or dependency name contains parentheses, it is assumed to be an archive library, the string within parentheses referring to a member within the library. Thus **lib(file.o)** and **\$(LIB)(file.o)** both refer to an archive library that contains **file.o**. (This assumes the **LIB** macro has been previously defined.) The expression **\$(LIB)(file1.o file2.o)** is not legal. Rules pertaining to archive libraries have the form **.XX.a** where the **XX** is the suffix from which the archive member is to be made.

An unfortunate byproduct of the current implementation requires the **XX** to be different from the suffix of the archive member. Thus, one cannot have **lib(file.o)** depend upon **file.o** explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib:
 lib(file1.o) lib(file2.o) lib(file3.o)
 @echo lib is now up-to-date
.c.a:
 $(CC) -c $(CFLAGS) $<
 $(AR) $(ARFLAGS) $@ $*.o
 rm -f $*.o
```

In fact, the **.c.a** rule listed above is built into *make* and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib:
 lib(file1.o) lib(file2.o) lib(file3.o)
 $(CC) -c $(CFLAGS) $(?:.o=.c)
 $(AR) $(ARFLAGS) lib $?
 rm $? @echo lib is now up-to-date
.c.a;;
```

Here the substitution mode of the macro expansions is used. The **\$?** list is defined to be the set of object filenames (inside **lib**) whose C source files are out-of-date. The substitution mode translates the **.o** to **.c**. (Unfortunately, one cannot as yet transform to **.c**; however, this may become possible in the future.) Note also, the disabling of the **.c.a**: rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

## Files

---

[Mm]akefile and s.[Mm]akefile  
/bin/sh

## See Also

---

cc(CP), lex(CP), yacc(CP), printf(S), sccsfile(F).  
cd(C), sh(C) in the *User's Reference*.

## Notes

---

Some commands return non-zero status inappropriately; use **-i** to overcome the difficulty.

Filenames with the characters **=**, **:**, or **@** will not work.

Commands that are directly executed by the shell, notably **cd(C)**, are ineffectual across new-lines in *make*.

The syntax **lib(file1.o file2.o file3.o)** is illegal.

You cannot build **lib(file.o)** from **file.o**.

The macro **\$(a:.o=.c)** does not work.

Named pipes are not handled well.

## Standards Conformance

---

*make* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



## **masm**

---

invokes the assembler

### **Syntax**

---

**masm** [ options ] sourcefile

### **Description**

---

*masm* is the 8086/286/386 assembler. It reads and assembles 8086/80286/80386 assembly language instructions from the source file named *sourcefile*. It then creates a linkable object file name *sourcefile.o*, or an executable program named **a.out**.

The extension *.s* is recommended but not required. If this extension is not given, *masm* displays a warning and continues processing.

There are the following options:

- **a**            This options puts the assembled output segments in alphabetic order before copying them to the object file.
- **c**            Outputs cross reference data for each assembled file to *filename.crf*.
- **C**            Outputs cross reference data for a set of assembled file. The cross reference data is written to files with the same names as the input files, with the filename extension ".erf."
- **d**            Adds a pass 1 listing to the assembly listing file *filename.lst*.
- **Dsym**        Defines the symbol appended to the **-D** flag as a null TEXTMACRO.
- **e**            Generates floating point code to emulate the 8087 or 287 coprocessor. Programs created with this option must be linked with an appropriate math library before being executed.
- **I**            *path* Defines the path appended to the **-I** flag as the search path for include files. Up to 10 include paths are allowed in one invocation of *masm*.
- **l**            [*listfile*] Creates an assembly listing file with the same basename as the *sourcefile* or, if the *listfile* parameter is given, with that name but with a ".lst" extension. The file

lists the source instructions, the assembled (binary code) for each instruction and any assembly errors. If filename is "-", the listing is written to *stdout*.

- **Mx** This option directs *masm* to preserve lower case letters in public and external names only when copying these names to the object file. For all other purposes, *masm* converts the lower case to upper case.
- **Mu** Disables case sensitivity. Upper case is now treated as identical to lower case.
- **MI** Leave case of symbols alone.
- **n** This option generates information about the symbols used in the assembled programs. The **-l** option must also be used for this option to take effect.
- **oobjfile** Copies the assembled instructions in octal to the file named *objfile*. This file is executable only if no errors occurred during the assembly. This option overrides the default object file name.
- **Oobjfile** Copies the assembled instructions in binary to the file named *objfile*.
- **r** Generates floating point code that can only be executed by an 8087 or 287 coprocessor.
- **v** Prints verbose error statistics on console. If not selected, only error counts are displayed.
- **x** displays error messages on the standard error channel, in addition to the messages generated in the listing file.
- **X** Copies to the assembly listing all statements forming the body of an IF directive whose expression (or condition) evaluates to false.

## Files

---

/bin/masm

## See Also

---

a.out(F), cc(CP), ld(CP)  
Macro Assembler User's Guide

## Notes

---

The default options are **-MI** and **-e** which enable case sensitivity and allow emulation of a floating point processor. The options are flags with the following default settings:

| Flag       | Default        | Meaning of TRUE condition            |
|------------|----------------|--------------------------------------|
| a          | FALSE          | Outputs segments alphabetically      |
| c          | FALSE          | Outputs cross reference data         |
| C          | FALSE          | Outputs cross reference data         |
| d          | FALSE          | Adds pass 1 listing to filename.lst  |
| Dsym       | NULL           | No meaning if not defined            |
| e          | FALSE          | Floating Point emulation             |
| I path     | NULL           | No meaning if not defined            |
| l listfile | sourcefile.lst | Sourcefile is the default filename   |
| M          | l              | Leave symbol case alone              |
| n          | TRUE           | Outputs symbols if -I selected       |
| o          | TRUE           | Assembled output in binary           |
| O          | FALSE          | Assembled output in octal            |
| r          | TRUE           | Real 8087 instead of emulated format |
| v          | FALSE          | Prints verbose error statistics      |
| x          | TRUE           | Displays errors on console           |
| X          | FALSE          | Toggle setting of conditional flag   |

## Return Value

---

The *masm* exit codes have the following meanings:

### Code Meaning

|   |                                                        |
|---|--------------------------------------------------------|
| 0 | No error                                               |
| 1 | Argument error                                         |
| 2 | Unable to open input file                              |
| 3 | Unable to open listing file                            |
| 4 | Unable to open object file                             |
| 5 | Unable to open cross reference file                    |
| 6 | Unable to open include file                            |
| 7 | Assembly errors. If fatal, the object file is deleted. |
| 8 | Memory allocation error                                |
| 9 | Real number input not allowed in this version.         |

## Value Added

---

*masm* is an extension of AT&T System V provided by the Santa Cruz Operation.



## mcs

---

manipulate the object file comment section

### Syntax

---

**mcs** [options] object-file ...

### Description

---

The *mcs* command manipulates the comment section, normally the ".comment" section, in an object file. It is used to add to, delete, print, and compress the contents of the comment section in a system object file. The *mcs* command must be given one or more of the options described below. It takes each of the options given and applies them in order to the *object-files*.

If the object file is an archive, the file is treated as a set of individual object files. For example, if the **-a** option is specified, the string is appended to the comment section of each archive element.

The following options are available.

**-a** *string*

Append *string* to the comment section of the *object-files*. If *string* contains embedded blanks, it must be enclosed in quotation marks.

**-c** Compress the contents of the comment section. All duplicate entries are removed. The ordering of the remaining entries is not disturbed.

**-d** Delete the contents of the comment section from the object file. The object file comment section header is removed also.

**-n** *name*

Specify the name of the section to access. By default, *mcs* deals with the section named *.comment*. This option can be used to specify another section.

**-p** Print the contents of the comment section on the standard output. If more than one name is specified, each entry printed is tagged by the name of the file from which it was extracted, using the format "filename:string."

## Examples

---

`mcs -p file`      # Print *file's comment* section.

`mcs -a string file` # Append *string* to *file's comment* section

## Files

---

`TMPPDIR/mcs*`      temporary files

`TMPPDIR/*`      temporary files

*TMPPDIR* is usually `/usr/tmp` but can be redefined by setting the environment variable **TMPPDIR** (see *tempnam()* in *tempnam(S)*).

## See Also

---

`cpp(CP)`, `a.out(F)`

## Notes

---

Note that this utility operates only on COFF executables.

The *mcs* command cannot add new sections or delete existing sections to executable objects with magic number 0413 (see *a.out(F)*).

## Value Added

---

*mcs* is an extension of AT&T System V provided by the Santa Cruz Operation.

# mkshlib

create a shared library

## Syntax

**mkshlib** -s *specfil* -t *target* [-h *host*] [-n] [-L *dir* ...] [-q]

## Description

*mkshlib* builds both the host and target shared libraries. A shared library is similar in function to a normal, non-shared library, except that programs that link with a shared library will share the library code during execution, whereas programs that link with a non-shared library will get their own copy of each library routine used.

The host shared library is an archive that is used to link-edit user programs with the shared library (see *ar*(F)). A host shared library can be treated exactly like a non-shared library and should be included on *cc*(CP) command lines in the usual way (see *cc*(CP)). Further, all operations that can be performed on an archive can also be performed on the host shared library.

The target shared library is an executable module that is bound into the user's address space during execution of a program using the shared library. The target shared library contains the code for all the routines in the library and must be fully resolved. The target will be brought into memory during execution of a program using the shared library, and subsequent processes that use the shared library will share the copy of code already in memory. The text of the target is always shared, but each process will get its own copy of the data.

The user interface to *mkshlib* consists of command line options and a shared library specification file. The shared library specification file describes the contents of the shared library.

The *mkshlib* command invokes other tools such as the archiver, *ar*(CP), the assembler, *as*(CP), and the link editor, *ld*(CP). Tools are invoked through the use of *execvp* (see *exec*(S)), which searches directories in the user's PATH. Also, prefixes to *mkshlib* are passed in the same manner as prefixes to the *cc*(CP) command, and invoked tools are given the prefix, where appropriate. For example, *i386mkshlib* will invoke *i386ld*.

The following command line options are recognized by *mkshlib*:

-s *specfil* Specifies the shared library specification file, *specfil*. This file contains the information necessary to build a shared library. Its contents include the branch table



specifications for the target, the path name in which the target should be installed, the start addresses of text and data for the target, the initialization specifications for the host, and the list of object files to be included in the shared library (see details below).

- t target** Specifies the output filename of the target shared library being created. It is assumed that this file will be installed on the target machine at the location given in the specification file (see the **#target** directive below). If the **-n** option is used, then a new target shared library will not be generated.
- h host** Specifies the output filename of the host shared library being created. If this option is not given, then the host shared library will not be produced.
- n** Do not generate a new target shared library. This option is useful when producing only a new host shared library. The **-t** option must still be supplied since a version of the target shared library is needed to build the host shared library.
- L dir ...** Change the algorithm of searching for the host shared libraries specified with the **#objects noload** directive to look in *dir* before looking in the default directories. The **-L** option can be specified multiple times on the command line in which case the directories given with the **-L** options are searched in the order given on the command line before the default directories.
- q** Quiet warning messages. This option is useful when warning messages are expected but not desired.

The shared library specification file contains all the information necessary to build both the host and target shared libraries. The contents and format of the specification file are given by the directives listed below.

All directives that can be followed by multi-line specifications are valid until the next directive or the end of the file.

#### **#address sectname address**

Specifies the start address, *address*, of section *sectname* for the target. This directive typically is used to specify the start addresses of the **.text** and **.data** sections. One **#address** per section name is valid. A **#address** directive must be given exactly once for the **.text** section and once for the **.data** section.

**#target *pathname***

Specifies the absolute path name, *pathname*, at which the target shared library will be installed on the target machine. The operating system uses this path name to locate the shared library when executing **a.out** files that use this shared library. This directive must be specified exactly once per specification file.

**#branch**

Specifies the start of the branch table specifications. The lines following this directive are taken to be branch table specification lines.

Branch table specification lines have the following format:

*funcname* <white space> *position*

where *funcname* is the name of the symbol given a branch table entry and *position* specifies the position of *funcname*'s branch table entry. *position* may be a single integer or a range of integers of the form *position1-position2*. Each *position* must be greater than or equal to one, the same position can not be specified more than once, and every position, from one to the highest given position, must be accounted for.

If a symbol is given more than one branch table entry by associating a range of positions with the symbol or by specifying the same symbol on more than one branch table specification line, then the symbol is defined to have the address of the highest associated branch table entry. All other branch table entries for the symbol can be thought of as "empty" slots and can be replaced by new entries in future versions of the shared library. Only functions should be given branch table entries, and those functions must be **external** symbols.

This directive must be specified exactly once per shared library specification file.

**#objects**

The lines following this directive are taken to be the list of input object files in the order they are to be loaded into the target. The list simply consists of each path name followed by a newline character. This list is also used to determine the input object files for the host shared library, but the order for the host is given by running the list through *lorder*(CP) and *tsort*(CP).

This directive must be specified exactly once per shared library specification file.



**#objects noload**

The **#objects noload** is followed by a list of host shared libraries. These libraries are searched in the order listed to resolve undefined symbols from the library being built. During the search it is considered an error if a non-shared version of a symbol is found before a shared version of the symbol.

Each name given is assumed to be a path name to a host or an argument of the form **-lX**, where **libX.a** is the name of a file in **LIBDIR** or **LLIBDIR**. This behavior is identical to that of *ld*, and the **-L** option can be used on the command line to specify other directories in which to locate these archives.

Note that if a host shared library is specified using **#objects noload**, any *cc* command that links to the shared library being built will need to specify that host also.

**#hide linker [\*]**

This directive changes symbols that are normally **external** into **static** symbols, local to the library being created. A regular expression may be given (*sh(C)*, *find(C)*), in which case all **external** symbols matching the regular expression are hidden; the **#export** directive (see below) can be used to counter this effect for specified symbols.

The optional “\*” is equivalent to the directive

```
#hide linker
*
```

and causes all **external** symbols to be made into **static** symbols.

All symbols specified in **#init** and **#branch** directives are assumed to be **external** symbols, and cannot be changed into **static** symbols using the **#hide** directive.

**#export linker [\*]**

Symbols given in the **#export** directive are **external** symbols (global among files) that, because of a regular expression in a **#hide** directive, would otherwise have been made **static**. For example,

```
#hide linker *
#export linker
one
two
```

causes all symbols except *one*, *two*, and those used in **#branch** and **#init** entries to be tagged as **static**.



**#init** *object*

Specifies that the object file, *object*, requires initialization code. The lines following this directive are taken to be initialization specification lines.

Initialization specification lines have the following format:

*ptr* <white space> *import*

*ptr* is a pointer to the associated imported symbol, *import*, and must be defined in the current specified object file, *object*. The initialization code generated for each such line is of the form:

*ptr* = &*import*;

All initializations for a particular object file must be given once and multiple specifications of the same object file are not allowed.

**#ident** *string*

Specifies a string, *string*, to be included in the .comment section of the target shared library.

**##**

Specifies a comment. All information on the line beginning with **##** is ignored.

## Files

---

*TEMPDIR*/\*      temporary files

*TEMPDIR* is usually /usr/tmp but can be redefined by setting the environment variable *TEMPDIR* [see *tempnam()* in *tempnam(S)*].

*LIBDIR*            usually /lib

*LLIBDIR*          usually /usr/lib

## See Also

---

ar(CP), as(CP), cc(CP), chkshlib(CP), ld(CP), lorder(CP), tsort(CP), a.out(F), ar(F)

## Notes

---

The **-n** option cannot be used with the **#objects noload** directive.

If *mkshlib* is asked to create a host library and a host of that name already exists, *mkshlib* will update the host using **ar -ru**. This means that you should always remove the host before rebuilding whenever an object file previously included in the library is removed or renamed.

If the address specified with the **#address** directive is outside user space, the library build may look successful, but if you try to use it, it might not work.

## mkstr

---

creates an error message file from C source

### Syntax

---

**mkstr** [-] messagefile prefix file ...

### Description

---

*mkstr* is used to create files of error messages. Its use can make programs with large numbers of error diagnostics much smaller, and reduce system overhead in running the program as the error messages do not have to be constantly swapped in and out.

*mkstr* will process each specified *file*, placing a massaged version of the input file in a file whose name consists of the specified *prefix* and the original name. The optional dash (-) causes the error messages to be placed at the end of the specified message file for recompiling part of a large *mkstr* ed program.

A typical *mkstr* command line is

```
mkstr pistrings xx *.c
```

This command causes all the error messages from the C source files in the current directory to be placed in the file *pistrings* and processed copies of the source for these files to be placed in files whose names are prefixed with *xx*.

To process the error messages in the source to the message file, *mkstr* keys on the string 'error("' in the input stream. Each time it occurs, the C string starting at the '"' is placed in the message file followed by a null character and a newline character; the null character terminates the message so it can be easily used when retrieved, the newline character makes it possible to sensibly *cat* the error message file to see its contents. The massaged copy of the input file then contains a *lseek* pointer into the file which can be used to retrieve the message. For example, the command changes

```
error("Error on reading", a2, a3, a4);

into

error(m, a2, a3, a4);
```



where *m* is the seek position of the string in the resulting error message file. The programmer must create a routine *error* which opens the message file, reads the string, and prints it out. The following example illustrates such a routine.

## Example

---

```
char efilename[] = "/usr/lib/pi_strings";
int efil = -1;

error(a1, a2, a3, a4)
int a1, a2, a3, a4;
{
 char buf[256];

 if (efil < 0) {
 efil = open(efilename, 0);
 if (efil < 0) {
 perror(efilename);
 exit(1);
 }
 }
 if (lseek(efil, (long) a1, 0) ||
 read(efil, buf, 256) <= 0) {
 printf("Unable to find error msg ");
 printf("at seek address %d\n", a1);
 exit(1);
 }
 printf(buf, a2, a3, a4);
}
```

## See Also

---

`lseek(S), xstr(CP)`

## Credit

---

This utility was developed at the University of California at Berkeley and is used with permission.

## Notes

---

All the arguments except the name of the file to be processed are unnecessary.

## nm

---

print name list of common object file

### Syntax

---

**nm [-oxhvnfurpVT] filename ...**

### Description

---

The *nm* command displays the symbol table of each common object file, *filename*. *Filename* may be a relocatable or absolute common object file; or it may be an archive of relocatable or absolute common object files. For each symbol, the following information will be printed:

|                |                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>    | The name of the symbol.                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Value</b>   | Its value expressed as an offset or an address depending on its storage class.                                                                                                                                                                                                                                                                                                                                                             |
| <b>Class</b>   | Its storage class.                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Type</b>    | Its type and derived type. If the symbol is an instance of a structure or of a union, then the structure or union tag will be given following the type (for example, struct-tag). If the symbol is an array, then the array dimensions will be given following the type (for example, char[ n ][ m ] ). Note that the object file must have been compiled with the -g option of the <i>cc</i> (CP) command for this information to appear. |
| <b>Size</b>    | Its size in bytes, if available. Note that the object file must have been compiled with the -g option of the <i>cc</i> (CP) command for this information to appear.                                                                                                                                                                                                                                                                        |
| <b>Line</b>    | The source line number at which it is defined, if available. Note that the object file must have been compiled with the -g option of the <i>cc</i> (CP) command for this information to appear.                                                                                                                                                                                                                                            |
| <b>Section</b> | For storage classes static and external, the object file section containing the symbol (for example, text, data, or bss).                                                                                                                                                                                                                                                                                                                  |

The output of *nm* may be controlled using the following options:

|           |                                                                   |
|-----------|-------------------------------------------------------------------|
| <b>-o</b> | Print the value and size of a symbol in octal instead of decimal. |
|-----------|-------------------------------------------------------------------|

- x** Print the value and size of a symbol in hexadecimal instead of decimal.
- h** Do not display the output header data.
- v** Sort external symbols by value before they are printed.
- n** Sort external symbols by name before they are printed.
- e** Print only external and static symbols.
- f** Produce full output. Print redundant symbols (.text, .data, .lib, and .bss), normally suppressed.
- u** Print undefined symbols only.
- r** Prepend the name of the object file or archive to each output line.
- p** Produce easily parsable, terse output. Each symbol name is preceded by its value (blanks if undefined) and one of the letters **U** (undefined), **A** (absolute), **T** (text segment symbol), **D** (data segment symbol), **S** (user-defined segment symbol), **R** (register symbol), **F** (file symbol), or **C** (common symbol). If the symbol is local (non-external), the type letter is in lowercase.
- V** Print the version of the nm command executing on the standard error output.
- T** By default, *nm* prints the entire name of the symbols listed. Since object files can have symbols names with an arbitrary number of characters, a name that is longer than the width of the column set aside for names will overflow its column, forcing every column after the name to be misaligned. The **-T** option causes *nm* to truncate every name which would otherwise overflow its column and place an asterisk as the last character in the displayed name to mark it as truncated.

Options may be used in any order, either singly or in combination, and may appear anywhere in the command line. Therefore, both **nm name** **-e -v** and **nm -ve name** print the static and external symbols in *name*, with external symbols sorted by value.



## Files

---

*TMPPDIR*/\* temporary files

*TMPPDIR* is usually */usr/tmp* but can be redefined by setting the environment variable **TMPPDIR** [see *tempnam()* in *tempnam(S)*].

## See Also

---

as(CP), cc(CP), ld(CP), tempnam(S), a.out(F), ar(F).

## Diagnostics

---

“nm: name: cannot open”  
if *name* cannot be read.

“nm: name: bad magic”  
if *name* is not a common object file.

“nm: name: no symbols”  
if the symbols have been stripped from *name*.

## Notes

---

If you are using XENIX binaries, please refer to the manual entry for this utility in the *XENIX Development Guide* for information on the appropriate usage with XENIX binaries.

When all the symbols are printed, they must be printed in the order they appear in the symbol table in order to preserve scoping information. Therefore, the **-v** and **-n** options should be used only in conjunction with the **-e** option.

## Standards Conformance

---

*nm* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## os2ld

### OS/2 cross linker

---

### Syntax

---

**os2ld** [ options ] file ...

### Description

---

*os2ld* links the object files(s) given by *file* to create a program for execution under OS/2. Although similar to *ld*(CP), *os2ld* has many options that differ significantly from *ld*. The options are described below:

- B** Produce a .COM file. This instructs *os2ld* to produce an executable .COM file.
- D** DS Allocate. This instructs *os2ld* to perform DS allocation. It is generally used in conjunction with the **-H** option.
- H** Load high. This option instructs *os2ld* to set a field in the header of the executable file to tell OS/2 to load the program at the highest available position in memory. It is most often used with programs in which data precedes code in the memory image.
- L** Include line numbers. This option instructs *os2ld* to include line numbers in the listing file (if any). Note that *os2ld* cannot put line numbers in the listing file if the source translator hasn't put them in the object file.
- M**  
Include public symbols. This option instructs *os2ld* to include public symbols in the list file. The symbols are sorted twice, lexicographically and by address.
- C** Ignore case. This option instructs *os2ld* to treat upper and lower case characters in symbol names as identical.
- F** *num*  
Set stack size. This option should be followed by a hexadecimal number. *os2ld* will use this number for the size in bytes of the stack segment in the output file.
- P** Pack code segments.
- Pc**[<dec#>]  
Pack code segments; the default limit for a segment is 64k minus 36.

**-Pd[<dec#>]**

Pack data segments; default limit is 64k.

**-Q** Don't pack code segments; this option is the default for segmented-executable pack code.**-S num**

Set segment limit. This option should be followed by a decimal number between 1 and 1024. The number sets the limit on the number of different segments that may be linked together. The default is 128. Note that the higher the value given, the slower the link will be.

**-m filename**

Create map file. This option should be followed by a filename. *os2ld* will create a file with the given name in which it will put information about the segments and groups in the executable. Additionally, public symbols and line numbers will be listed in this file if the **-M** and **-L** options are given.

**-nl num**

Set name length. This option should be followed by a decimal number. The option instructs *os2ld* to truncate all public and external symbols longer than *num* characters.

**-o filename**

Name output file. This option should be followed by a filename which *os2ld* will use as the name of the executable file it creates. The default name is **a.out**.

**-u name**

Name undefined symbol. This option should be followed by a symbol name. *os2ld* will enter the given name into its symbol table as an undefined symbol. The **-u** option may appear more than once on the command line.

As with *ld*, the files passed to *os2ld* may be either UNIX-style libraries (objects collected using *ar*(CP) and indexed using *ranlib*(CP)) or ordinary 8086 object files. Unless the **-u** option appears, at least one of the files passed to *os2ld* must be an ordinary object file. Libraries are searched only after all the ordinary object files have been processed.

## Files

---

/usr/bin/os2ld

## See Also

---

*ar*(CP), *as*(CP), *cc*(CP), *dosld*(CP), *ld*(CP), *ranlib*(CP)



## **Value Added**

---

*os2ld* is an extension of AT&T System V provided by the Santa Cruz Operation.

## prof

---

display profile data

### Syntax

---

**prof** [-tcan] [-ox] [-g] [-z] [-h] [-s] [-m mdata] [prog]

### Description

---

The *prof* command interprets a profile file produced by the *monitor*(S) function. The symbol table in the object file *prog* (**a.out** by default) is read and correlated with a profile file (**mon.out** by default). For each external text symbol the percentage of time spent executing between the address of that symbol and the address of the next is printed, together with the number of times that function was called and the average number of milliseconds per call.

The mutually exclusive options **t**, **c**, **a**, and **n** determine the type of sorting of the output lines:

- t Sort by decreasing percentage of total time (default).
- c Sort by decreasing number of calls.
- a Sort by increasing symbol address.
- n Sort lexically by symbol name.

The mutually exclusive options **o** and **x** specify the printing of the address of each symbol monitored:

- o Print each symbol address (in octal) along with the symbol name.
- x Print each symbol address (in hexadecimal) along with the symbol name.

The following options may be used in any combination:

- g Include non-global symbols (static functions).
- z Include all symbols in the profile range (see *monitor*(S)), even if associated with zero number of calls and zero time.
- h Suppress the heading normally printed on the report. (This is useful if the report is to be processed further.)

-s Print a summary of several of the monitoring parameters and statistics on the standard error output.

-m *mdata*

Use file *mdata* instead of **mon.out** as the input profile file.

A program creates a profile file if it has been loaded with the -p option of *cc*(CP). This option to the *cc* command arranges for calls to *monitor*(S) at the beginning and end of execution. It is the call to *monitor* at the end of execution that causes a profile file to be written. The number of calls to a function is tallied if the -p option was used when the file containing the function was compiled.

The name of the file created by a profiled program is controlled by the environment variable *PROFDIR*. If *PROFDIR* does not exist, "mon.out" is produced in the directory that is current when the program terminates. If *PROFDIR* = string, "string/pid.progname" is produced, where *progname* consists of *argv*[0] with any path prefix removed, and *pid* is the program's process id. If *PROFDIR* is the null string, no profiling output is produced.

A single function may be split into subfunctions for profiling by means of the *MARK* macro (see *prof*(M)).

## Files

---

mon.out for profile  
a.out for namelist

## See Also

---

*cc*(CP), *exit*(S), *profil*(S), *monitor*(S), *prof*(M).

## Warning

---

The times reported in successive identical runs may show variances of 20% or more, because of varying cache-hit ratios due to sharing of the cache with other processes. Even if a program seems to be the only one using the machine, hidden background or asynchronous processes may blur the data. In rare cases, the clock ticks initiating recording of the program counter may "beat" with loops in a program, grossly distorting measurements.

Call counts are always recorded precisely.

The times for static functions are attributed to the preceding external text symbol if the -g option is not used. However, the call counts for the preceding function are still correct (that is, the static function call counts are not added in with the call counts of the external function).



## Notes

---

If you are using XENIX binaries, please refer to the manual entry for this utility in the *XENIX Development Guide* for information on the appropriate usage with XENIX binaries.

Only programs that call *exit*(S) or return from *main* will cause a profile file to be produced, unless a final call to monitor is explicitly coded.

The use of the **-p** option to *cc*(CP) to invoke profiling imposes a limit of 600 functions that may have call counters established during program execution. For more counters you must call *monitor*(S) directly. If this limit is exceeded, other data will be overwritten and the **mon.out** file will be corrupted. The number of call counters used will be reported automatically by the *prof* command whenever the number exceeds 5/6 of the maximum.

## Standards Conformance

---

*prof* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## prs

---

print an SCCS file

### Syntax

---

**prs** [-d[dataspec]] [-r[SID]] [-e] [-l] [-c[date-time]] [-a] files

### Description

---

The *prs* command prints, on the standard output, parts or all of an SCCS file (see *sccsfile* (F)) in a user-supplied format. If a directory is named, *prs* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

Arguments to *prs*, which may appear in any order, consist of *keyletter* arguments and file names.

All the described *keyletter* arguments apply independently to each named file:

- |                     |                                                                                                                                                                                                        |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-d[dataspec]</b> | Used to specify the output data specification. The <i>dataspec</i> is a string consisting of SCCS file <i>data keywords</i> (see <i>DATA KEYWORDS</i> ) interspersed with optional user-supplied text. |
| <b>-r[SID]</b>      | Used to specify the SCCS identification (SID) string of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed.                       |
| <b>-e</b>           | Requests information for all deltas created <i>earlier</i> than and including the delta designated via the <b>-r</b> keyletter or the date given by the <b>-c</b> option.                              |
| <b>-l</b>           | Requests information for all deltas created <i>later</i> than and including the delta designated via the <b>-r</b> keyletter or the date given by the <b>-c</b> option.                                |

**-c[*date-time*]**The cutoff date-time **-c[cutoff]** is in the form:

YY[MM[DD[HH[MM[SS]]]]]

Units omitted from the date-time default to their maximum possible values; that is, **-c7502** is equivalent to **-c750228235959**. Any number of non-numeric characters may separate the various 2-digit pieces of the *cutoff* date in the form: **"-c77/2/2 9:22:25"**. The **-c** option must be used with the **-e** or **-l** option.

**-a**

Requests printing of information for both removed, i.e., delta type = *R*, (see *rmdel*(CP)) and existing, i.e., delta type = *D*, deltas. If the **-a** keyletter is not specified, information for existing deltas only is provided.

## Data Keywords

---

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see *sccsfile*(F)) have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by *prs* consists of: (C) the user-supplied text; and (S) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multiline* (M), in which keyword substitution is followed by a carriage return.

User-supplied text is any text other than recognized data keywords. A tab is specified by `\t` and carriage return/new-line is specified by `\n`. The default data keywords are:

```
" :Dt:t:DL:\nMRs:\n:MR:COMMENTS:\n:C:"
```



TABLE 1. SCCS Files Data Keywords

| Keyword | Data Item                               | File Section | Value           | Format |
|---------|-----------------------------------------|--------------|-----------------|--------|
| :Dt:    | Delta information                       | Delta Table  | See below*      | S      |
| :DL:    | Delta line statistics                   | "            | :Li:/:Ld:/:Lu:  | S      |
| :Li:    | Lines inserted by Delta                 | "            | nnnnn           | S      |
| :Ld:    | Lines deleted by Delta                  | "            | nnnnn           | S      |
| :Lu:    | Lines unchanged by Delta                | "            | nnnnn           | S      |
| :DT:    | Delta type                              | "            | D~or~R          | S      |
| :I:     | SCCS ID string (SID)                    | "            | :R:/:L:/:B:/:S: | S      |
| :R:     | Release number                          | "            | nnnn            | S      |
| :L:     | Level number                            | "            | nnnn            | S      |
| :B:     | Branch number                           | "            | nnnn            | S      |
| :S:     | Sequence number                         | "            | nnnn            | S      |
| :D:     | Date Delta created                      | "            | :Dy:/:Dm:/:Dd:  | S      |
| :Dy:    | Year Delta created                      | "            | nn              | S      |
| :Dm:    | Month Delta created                     | "            | nn              | S      |
| :Dd:    | Day Delta created                       | "            | nn              | S      |
| :T:     | Time Delta created                      | "            | :Th:/:Tm:/:Ts:  | S      |
| :Th:    | Hour Delta created                      | "            | nn              | S      |
| :Tm:    | Minutes Delta created                   | "            | nn              | S      |
| :Ts:    | Seconds Delta created                   | "            | nn              | S      |
| :P:     | Programmer who created Delta            | "            | logname         | S      |
| :DS:    | Delta sequence number                   | "            | nnnn            | S      |
| :DP:    | Predecessor Delta seq-no.               | "            | nnnn            | S      |
| :DI:    | Seq-no. of deltas incl., excl., ignored | "            | :Dn:/:Dx:/:Dg:  | S      |
| :Dn:    | Deltas included (seq #)                 | "            | :DS:~:DS: ...   | S      |
| :Dx:    | Deltas excluded (seq #)                 | "            | :DS:~:DS: ...   | S      |
| :Dg:    | Deltas ignored (seq #)                  | "            | :DS:~:DS: ...   | S      |
| :MR:    | MR numbers for delta                    | "            | text            | M      |
| :C:     | Comments for delta                      | "            | text            | M      |
| :UN:    | User names                              | User Names   | text            | M      |
| :FL:    | Flag list                               | Flags        | text            | M      |
| :Y:     | Module type flag                        | "            | text            | S      |
| :MF:    | MR validation flag                      | "            | yes~or~no       | S      |
| :MP:    | MR validation pgm name                  | "            | text            | S      |
| :KF:    | Keyword error/warning flag              | "            | yes~or~no       | S      |
| :KV:    | Keyword validation string               | "            | text            | S      |
| :BF:    | Branch flag                             | "            | yes~or~no       | S      |
| :J:     | Joint edit flag                         | "            | yes~or~no       | S      |
| :LK:    | Locked releases                         | "            | :R: ...         | S      |
| :Q:     | User-defined keyword                    | "            | text            | S      |
| :M:     | Module name                             | "            | text            | S      |

(Continued on next page)

TABLE 1. SCCS Files Data Keywords (*Continued.*)

| Keyword | Data Item                         | File Section | Value             | Format |
|---------|-----------------------------------|--------------|-------------------|--------|
| :FB:    | Floor boundary                    | "            | :R:               | S      |
| :CB:    | Ceiling boundary                  | "            | :R:               | S      |
| :Ds:    | Default SID                       | "            | :I:               | S      |
| :ND:    | Null delta flag                   | "            | yes~or~no         | S      |
| :FD:    | File descriptive text             | Comments     | text              | M      |
| :BD:    | Body                              | Body         | text              | M      |
| :GB:    | Gotten body                       | "            | text              | M      |
| :W:     | A form of <i>what</i> (CP) string | N/A          | :Z::M:t:I:        | S      |
| :A:     | A form of <i>what</i> (CP) string | N/A          | :Z::Y::~M::~I::Z: | S      |
| :Z:     | <i>what</i> (C) string delimiter  | N/A          | @(#)              | S      |
| :F:     | SCCS file name                    | N/A          | text              | S      |
| :PN:    | SCCS file path name               | N/A          | text              | S      |

\* :Dt::~DT::~I::~D::~T::~P::~DS::~DP:

## Examples

```
prs -d"Users and/or user IDs for :F: are:\n:UN:" s.file
```

may produce on the standard output:

Users and/or user IDs for s.file are:

xyz

131

abc

```
prs -d"Newest delta for pgm :M:: :I: Created :D: By :P:" -r s.file
```

may produce on the standard output:

Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas

As a special case:

```
prs s.file
```

may produce on the standard output:

D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000

MRs:

bl78-12345

bl79-54321

COMMENTS:

this is the comment line for s.file initial delta

for each delta table entry of the "D" type. The only keyletter argument allowed to be used with the *special case* is the *-a* keyletter.

## Files

---

/tmp/pr????

## See Also

---

admin(CP), delta(CP), get(CP), sccsfile(F)

## Standards Conformance

---

*prs* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

and The X/Open Portability Guide II of January 1987.



## ranlib

---

converts archives to random libraries.

### Syntax

---

**ranlib** archive...

### Description

---

*ranlib* converts each *archive* to a form which can be loaded more rapidly by the loader, by adding a table of contents named `__SYM-DEF` to the beginning of the archive. It uses *ar*(CP) to reconstruct the archive, so sufficient temporary file space must be available in the file system containing the current directory.

### See Also

---

*ld*(CP), *ar*(CP), *copy*(C), *settime*(ADM)

### Notes

---

Failure to process a library with *ranlib*, or failure to reprocess a library with *ranlib*, will cause *ld* to fail. Because generation of a library by *ar* and randomization by *ranlib* are separate, phase errors are possible. The loader *ld* warns when the modification date of a library is more recent than the creation of its dictionary; but this means you get the warning even if you only copy the library.

### Value Added

---

*ranlib* is an extension of AT&T System V provided by the Santa Cruz Operation.

## **rcc**

### **AT&T C compiler**

---

### **Syntax**

---

**rcc** [ options ] files

### **Description**

---

The *rcc* command invokes the AT&T C compiler. Note that this is a different compiler from the Microsoft C compiler that is available through *cc*(CP).

The AT&T compilation tools consist of a preprocessor, compiler, optimizer, assembler, and link editor. The *rcc* command processes the supplied options and then executes the various tools with the proper arguments. The *rcc* command accepts several types of files as arguments.

Files whose names end with *.c* are taken to be C source programs and may be preprocessed, compiled, optimized, assembled, and link edited. The compilation process may be stopped after the completion of any pass if the appropriate options are supplied. If the compilation process runs through the assembler, then an object program is produced and is left in the file whose name is that of the source with *.o* substituted for *.c*. However, the *.o* file is normally deleted if a single C program is compiled and then immediately link edited. In the same way, files whose names end in *.s* are taken to be assembly source programs and may be assembled and link edited; and files whose names end in *.i* are taken to be preprocessed C source programs and may be compiled, optimized, assembled, and link edited. Files whose names do not end in *.c*, *.s*, or *.i* are handed to the link editor.

Since the *rcc* command usually creates files in the current directory during the compilation process, it is necessary to run the *rcc* command in a directory in which a file can be created.

The following options are interpreted by *rcc*:

- c Suppress the link editing phase of the compilation and do not remove any produced object files.
- ds  
Do not generate symbol attribute information for the symbolic debugger.

**-dl**

Do not generate symbolic debugging line number information. This and the above flag may be used in conjunction as **-dsl** (**-dsl** is the default unless the **-g** flag is given).

**-g**

Cause the compiler to generate additional information needed for the use of *sdb*(CP).

**-o outfile**

Produce an output object file by the name *outfile*. The name of the default file is **a.out**. This is a link editor option.

**-p**

Arrange for the compiler to produce code that counts the number of times each routine is called; also, if link editing takes place, profiled versions of **libc.a** and **libm.a** (with **-lm** option) are linked and *monitor*(S) is automatically called. A **mon.out** file will then be produced at normal termination of execution of the object program. An execution profile can then be generated by use of *prof*(CP).

**-qp**

Arrange for profiled code to be produced where the **p** argument produces identical results to the **-p** option [allows profiling with *prof*(CP)].

**-E**

Run only *cpp*(CP) on the named C programs, and send the result to the standard output.

**-H**

Print out on *stderr* the path name of each file included during the current compilation.

**-O**

Do compilation phase optimization. This option will not have any effect on *.s* files.

**-P**

Run only *cpp*(CP) on the named C programs and leave the result in corresponding files suffixed *.i*. This option is passed to *cpp*(CP).

**-S**

Compile and do not assemble the named C programs, and leave the assembler-language output in corresponding files suffixed *.s*.

**-V**

Print the version of the compiler, optimizer, assembler and/or link editor that is invoked.

**-Wc,arg1[,arg2...]**

Hand off the argument[s] *argi* to pass *c* where *c* is one of [**p02al**] indicating the preprocessor, compiler, optimizer, assembler, or link editor, respectively. For example: **-Wa,-m** passes **-m** to the assembler.

**-Y [p02alSILU],dirname**

Specify a new path name, *dirname*, for the locations of the tools and directories designated in the first argument. [**p02alSILU**] represents:



**p** preprocessor  
**0** compiler  
**2** optimizer  
**a** assembler  
**l** link editor  
**S** directory containing the start-up routines  
**I** default include directory searched by *cpp*(CP)  
**L** first default library directory searched by *ld*(CP)  
**U** second default library directory searched by *ld*(CP)

If the location of a tool is being specified, then the new path name for the tool will be *dirname/tool*. If more than one **-Y** option is applied to any one tool or directory, then the last occurrence holds.

### **-Zp[1|2|4]**

Packs structure members in memory. Normally, structure members are aligned as follows: items of type **char** are byte-aligned, items of type **short** are aligned on two-byte boundaries, and all other types of structure members are word-aligned.

Specifying an option to **-Zp** will force alignment on the given byte boundary. If no option is used with **-Zp**, structure members will be packed on one-byte boundaries. The alignment may be altered with the **#pragma pack** preprocessor directive.

The *rcc* command also recognizes **-C**, **-D**, **-I**, and **-U** and passes these options and their arguments directly to the preprocessor without using the **-W** option. Similarly, the *rcc* command recognizes **-a**, **-l**, **-m**, **-r**, **-s**, **-t**, **-u**, **-x**, **-z**, **-L**, **-M**, and **-V** and passes these options and their arguments directly to the loader. See the manual pages for *cpp*(CP) and *ld*(CP) for descriptions.

Other arguments are taken to be C compatible object programs, typically produced by an earlier *rcc* run, or perhaps libraries of C compatible routines and are passed directly to the link editor. These programs, together with the results of any compilations specified, are link edited (in the order given) to produce an executable program with name **a.out** unless the **-o** option of the link editor is used.

If the *rcc* command is put in a file *prefixcc* the prefix will be parsed off the command and used to call the tools, i.e., *prefixtool*. For example, *OLDrcc* will call *OLDcpp*, *OLDcomp*, *OLDoptim*, *OLDas*, and *OLDld* and will link *OLDcrt1.o*. Therefore, one **MUST** be careful when moving the *rcc* command around. The prefix will apply to the preprocessor, compiler, optimizer, assembler, link editor, and the start-up routines.

The C language standard was extended to allow arbitrary length variable names. The option pair **“-Wp,-T -W0,-XT”** will cause *rcc* to truncate arbitrary length variable names.

## Files

---

|                              |                               |
|------------------------------|-------------------------------|
| <code>file.c</code>          | C source file                 |
| <code>file.i</code>          | preprocessed C source file    |
| <code>file.o</code>          | object file                   |
| <code>file.s</code>          | assembly language file        |
| <code>a.out</code>           | link edited output            |
| <code>LIBDIR/*rcrt1.o</code> | start-up routine              |
| <code>LIBDIR/rcrtn.o</code>  | start-up routine              |
| <code>TMPDIR/*</code>        | temporary files               |
| <code>LIBDIR/rcpp</code>     | preprocessor, <i>cpp</i> (CP) |
| <code>LIBDIR/rcomp</code>    | compiler                      |
| <code>LIBDIR/roptim</code>   | optimizer                     |
| <code>BINDIR/as</code>       | assembler, <i>as</i> (CP)     |
| <code>BINDIR/ld</code>       | link editor, <i>ld</i> (CP)   |
| <code>LIBDIR/</code>         | standard C library            |
| <code>LIBDIR/libc_s.a</code> | standard C shared library     |

*LIBDIR* is usually **/lib**.

*BINDIR* is usually **/bin**.

*TMPDIR* is usually **/usr/tmp** but can be redefined by setting the environment variable **TMPDIR** [see *tempnam()* in *tempnam(3S)*].

## See Also

---

*as*(CP), *ld*(CP), *cpp*(CP), *gcc*(CP), *lint*(CP), *prof*(CP), *sdb*(CP), *tmpnam*(S).

## Diagnostics

---

The diagnostics produced by the C compiler are sometimes cryptic.

## Notes

---

By default, the return value from a compiled C program is completely random. The only two guaranteed ways to return a specific value is to explicitly call *exit*(S) or to leave the function **main()** with a “*return expression;*” construct.

## regcmp

---

regular expression compile

### Syntax

---

**regcmp** [ - ] files

### Description

---

The *regcmp* command performs a function similar to *regcmp*(S) and, in most cases, precludes the need for calling *regcmp*(S) from C programs. This saves on both execution time and program size. The command *regcmp* compiles the regular expressions in *file* and places the output in *file.i*. If the - option is used, the output will be placed in *file.c*. The format of entries in *file* is a name (C variable) followed by one or more blanks followed by a regular expression enclosed in double quotes. The output of *regcmp* is C source code. Compiled regular expressions are represented as **extern char** vectors. *File.i* files may thus be *included* in C programs, or *file.c* files may be compiled and later loaded. In the C program which uses the *regcmp* output, *regex(abc,line)* will apply the regular expression named *abc* to *line*. Diagnostics are self-explanatory.

### Examples

---

```
name "[A-Za-z][A-Za-z0-9_]*"$0"
```

```
telno "\{0,1\}([2-9][01][1-9])$0\){0,1} *"
 "([2-9][0-9]{2})$1[-]{0,1}"
 "([0-9]{4})$2"
```

In the C program that uses the *regcmp* output,

```
regex(telno, line, area, exch, rest)
```

will apply the regular expression named *telno* to *line*.

### See Also

---

*regcmp*(S).



## rmDEL

---

remove a delta from an SCCS file

### Syntax

---

**rmDEL** -rSID files

### Description

---

The *rmDEL* command removes the delta specified by the *SID* from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the delta specified must *not* be that of a version being edited for the purpose of making a delta (i. e., if a *p-file* (see *get(CP)*) exists for the named SCCS file, the delta specified must *not* appear in any entry of the *p-file*).

The **-r** option is used for specifying the *SID* (SCCS identification) level of the delta to be removed.

If a directory is named, *rmDEL* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with *s.*) and unreadable files are silently ignored. If a name of **-** is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored.

Simply stated, the rules are:

(C)  
if you make a delta, you can remove it.

or

(S)  
if you own the file and directory, you can remove a delta.

### Files

---

|        |                         |
|--------|-------------------------|
| x.file | (see <i>delta(CP)</i> ) |
| z.file | (see <i>delta(CP)</i> ) |

### See Also

---

*delta(CP)*, *get(CP)*, *prs(CP)*, *scsfile(F)*.

## **Standards Conformance**

---

*rmDEL* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## sact

---

print current SCCS file editing activity

## Syntax

---

sact files

## Description

---

The *sact* command informs the user of any impending deltas to a named SCCS file. This situation occurs when *get*(CP) with the *-e* option has been previously executed without a subsequent execution of *delta*(CP). If a directory is named on the command line, *sact* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of - is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

The output for each named file consists of five fields separated by spaces.

|         |                                                                                                                          |
|---------|--------------------------------------------------------------------------------------------------------------------------|
| Field 1 | specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta. |
| Field 2 | specifies the SID for the new delta to be created.                                                                       |
| Field 3 | contains the logname of the user who will make the delta (i.e., executed a <i>get</i> for editing).                      |
| Field 4 | contains the date that <i>get -e</i> was executed.                                                                       |
| Field 5 | contains the time that <i>get -e</i> was executed.                                                                       |

## See Also

---

*delta*(CP), *get*(CP), *unget*(CP).

## Standards Conformance

---

*sact* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



## sccsdiff

---

compare two versions of an SCCS file

### Syntax

---

**sccsdiff** -rSID1 -rSID2 [-p] [-sn] files

### Description

---

The *sccsdiff* command compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but arguments apply to all files.

- |               |                                                                                                                                                    |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-rSID?</b> | <i>SID1</i> and <i>SID2</i> specify the deltas of an SCCS file that are to be compared. Versions are passed to <i>bdiff(C)</i> in the order given. |
| <b>-p</b>     | pipe output for each file through <i>pr(C)</i> .                                                                                                   |
| <b>-sn</b>    | <i>n</i> is the file segment size that <i>bdiff</i> will pass to <i>diff(C)</i> . This is useful when <i>diff</i> fails due to a high system load. |

### Files

---

/tmp/get????? Temporary files

### See Also

---

*get(CP)*.  
*bdiff(C)*, *pr(C)* in the *User's Reference*.

### Diagnostics

---

“*file*: No differences”

If the two versions are the same.

## sdb

### symbolic debugger

## Syntax

**sdb** [-w] [-W] [objfil [corfil [directory-list]]]

## Description

The *sdb* command calls a symbolic debugger that can be used with C programs. It may be used to examine their object files and core files and to provide a controlled environment for their execution.

*objfil* is a COFF or x.out format executable program file which has been compiled with the **-g** (debug) option. If it has not been compiled with the **-g** option, the symbolic capabilities of *sdb* will be limited, but the file can still be examined and the program debugged. The default for *objfil* is **a.out**. *corfil* is assumed to be a core image file produced after executing *objfil*; the default for *corfil* is **core**. The core file need not be present. A **-** in place of *corfil* will force *sdb* to ignore any core image file. The colon-separated list of directories (*directory-list*) is used to locate the source files used to build *objfil*.

It is useful to know that at any time there is a *current line* and *current file*. If *corfil* exists, then they are initially set to the line and file containing the source statement at which the process terminated. Otherwise, they are set to the first line in *main()*. The current line and file may be changed with the source file examination commands.

By default, warnings are provided if the source files used in producing *objfil* cannot be found, or are newer than *objfil*. This checking feature and the accompanying warnings may be disabled by the use of the **-W** flag.

Names of variables are written just as they are in C. *sdb* does not truncate names. Variables local to a procedure may be accessed using the form *procedure:variable*. If no procedure name is given, the procedure containing the current line is used by default.

It is also possible to refer to structure members as *variable.member*, pointers to structure members as *variable->member*, and array elements as *variable[number]*. Pointers may be dereferenced by using the form *pointer[0]*. Combinations of these forms may also be used. A number may be used in place of a structure variable name, in which case the number is viewed as the address of the structure, and the template used for the structure is that of the last structure referenced by *sdb*. An unqualified structure variable may also be used with various commands. Generally, *sdb* will interpret a structure as a set of



variables. Thus, *sdb* will display the values of all the elements of a structure when it is requested to display a structure. An exception to this interpretation occurs when displaying variable addresses. An entire structure does have an address, and it is this value *sdb* displays, not the addresses of individual elements.

Elements of a multidimensional array may be referenced as *variable [number][number]...*, or as *variable [number,number,...]*. In place of *number*, the form *number;number* may be used to indicate a range of values, \* may be used to indicate all legitimate values for that subscript, or subscripts may be omitted entirely if they are the last subscripts and the full range of values is desired. As with structures, *sdb* displays all the values of an array or of the section of an array if trailing subscripts are omitted. It displays only the address of the array itself or of the section specified by the user if subscripts are omitted.

A particular instance of a variable on the stack may be referenced by using the form *procedure:variable,number*. All the variations mentioned in naming variables may be used. *Number* is the occurrence of the specified procedure on the stack, counting the top, or most current, as the first. If no procedure is specified, the procedure currently executing is used by default.

It is also possible to specify a variable by its address. All forms of integer constants which are valid in C may be used, so that addresses may be input in decimal, octal, or hexadecimal.

Line numbers in the source program are referred to as *file-name: number* or *procedure: number*. In either case the number is relative to the beginning of the file. If no procedure or file name is given, the current file is used by default. If no number is given, the first line of the named procedure or file is used.

While a process is running under *sdb*, all addresses refer to the executing program; otherwise they refer to *objfil* or *corfil*. An initial argument of *-w* permits overwriting locations in *objfil*.

### Addresses

The address in a file associated with a written address is determined by a mapping associated with that file. Each mapping is represented by two triples (*b1, e1, f1*) and (*b2, e2, f2*) and the *file address* corresponding to a written *address* is calculated as follows:

```

b1 ≤ address < e1
then
file address = address + f1 - b1

```



otherwise

```

 b2 ≤ address < e2
 then
 file address = address + f2 - b2

```

otherwise, the requested *address* is not legal. In some cases (e.g., for programs with separated I and D space) the two segments for a file may overlap.

The initial setting of both mappings is suitable for normal **a.out** and **core** files. If either file is not of the kind expected, then, for that file, *b1* is set to 0, *e1* is set to the maximum file size, and *f1* is set to 0; in this way the whole file can be examined with no address translation.

In order for *sdb* to be used on large files, all appropriate values are kept as signed 32-bit integers.

### Commands

The commands for examining data in the program are:

- t**     Print a stack trace of the terminated or halted program.
- T**     Print the top line of the stack trace.

### *variable/clm*

Print the value of *variable* according to length *l* and format *m*. A numeric count *c* indicates that a region of memory, beginning at the address implied by *variable*, is to be displayed. The length specifiers are:

- b**   one byte
- h**   two bytes (half word)
- l**   four bytes (long word)

Legal values for *m* are:

- c**   character
- d**   decimal
- u**   decimal, unsigned
- o**   octal
- x**   hexadecimal
- f**   32-bit single precision floating point
- g**   64-bit double precision floating point
- s**   Assume *variable* is a string pointer and print characters starting at the address pointed to by the variable.

- a Print characters starting at the variable's address. This format may not be used with register variables.
- p pointer to procedure
- i disassemble machine-language instruction with addresses printed numerically and symbolically.
- I disassemble machine-language instruction with addresses just printed numerically.

Length specifiers are only effective with the **c**, **d**, **u**, **o**, and **x** formats. Any of the specifiers, *c*, *l*, and *m*, may be omitted. If all are omitted, *sdb* chooses a length and a format suitable for the variable's type as declared in the program. If *m* is specified, then this format is used for displaying the variable. A length specifier determines the output length of the value to be displayed, sometimes resulting in truncation. A count specifier *c* tells *sdb* to display that many units of memory, beginning at the address of *variable*. The number of bytes in one such unit of memory is determined by the length specifier *l*, or if no length is given, by the size associated with the *variable*. If a count specifier is used for the **s** or **a** command, then that many characters are printed. Otherwise successive characters are printed until either a null byte is reached or 128 characters are printed. The last variable may be redisplayed with the command *J*.

The *sh*(C) metacharacters **\*** and **?** may be used within procedure and variable names, providing a limited form of pattern matching. If no procedure name is given, variables local to the current procedure and global variables are matched; if a procedure name is specified, then only variables local to that procedure are matched. To match only global variables, the form *:pattern* is used.

*linenumber?lm*

*variable:?lm*

Print the value at the address from **a.out** or **I** space given by *linenumber* or *variable* (procedure name), according to the format *lm*. The default format is 'i'.

*variable=lm*

*linenumber=lm*

*number=lm*

Print the address of *variable* or *linenumber*, or the value of *number*, in the format specified by *lm*. If no format is given, then **lx** is used. The last variant of this command provides a convenient way to convert between decimal, octal, and hexadecimal.

*variable!value*

Set *variable* to the given *value*. The value may be a number, a character constant, or a variable. The value must be well defined; expressions which produce more than one value, such as structures, are not allowed. Character constants are denoted



'character. Numbers are viewed as integers unless a decimal point or exponent is used. In this case, they are treated as having the type double. Registers are viewed as integers. The *variable* may be an expression which indicates more than one variable, such as an array or structure name. If the address of a variable is given, it is regarded as the address of a variable of type *int*. C conventions are used in any type conversions necessary to perform the indicated assignment.

- x Print the machine registers and the current machine-language instruction.
- X Print the current machine-language instruction.

The commands for examining source files are:

e *procedure*  
 e *file-name*  
 e *directory/*  
 e *directory file-name*

The first two forms set the current file to the file containing *procedure* or to *file-name*. The current line is set to the first line in the named procedure or file. Source files are assumed to be in *directory*. The default is the current working directory. The latter two forms change the value of *directory*. If no procedure, file name, or directory is given, the current procedure name and file name are reported.

/*regular expression*/

Search forward from the current line for a line containing a string matching *regular expression* as in *ed(C)*. The trailing / may be deleted.

?*regular expression*?

Search backward from the current line for a line containing a string matching *regular expression* as in *ed(C)*. The trailing ? may be deleted.

- p Print the current line.
- z Print the current line followed by the next 9 lines. Set the current line to the last line printed.
- w Window. Print the 10 lines around the current line.

*number*

Set the current line to the given line number. Print the new current line.

*count*+

Advance the current line by *count* lines. Print the new current line.



*count*—

Retreat the current line by *count* lines. Print the new current line.

The commands for controlling the execution of the source program are:

*count* **r** *args*

*count* **R**

Run the program with the given arguments. The **r** command with no arguments reuses the previous arguments to the program while the **R** command runs the program with no arguments. An argument beginning with < or > causes redirection for the standard input or output, respectively. If *count* is given, it specifies the number of breakpoints to be ignored.

*linenumber* **c** *count*

*linenumber* **C** *count*

Continue after a breakpoint or interrupt. If *count* is given, the program will stop when *count* breakpoints have been encountered. The signal which caused the program to stop is reactivated with the **C** command and ignored with the **c** command. If a line number is specified, then a temporary breakpoint is placed at the line and execution is continued. The breakpoint is deleted when the command finishes.

*linenumber* **g** *count*

Continue after a breakpoint with execution resumed at the given line. If *count* is given, it specifies the number of breakpoints to be ignored.

**s** *count*

**S** *count*

Single-step the program through *count* lines. If no *count* is given, then the program is run for one line. **S** is equivalent to **s** except it steps through procedure calls.

**i**

**I**

Single-step by one machine-language instruction. The signal which caused the program to stop is reactivated with the **I** command and ignored with the **i** command.

*variable***\$m** *count*

*address***:m** *count*

Single-step (as with **s**) until the specified location is modified with a new value. If *count* is omitted, it is effectively infinity. *Variable* must be accessible from the current procedure. Since this command is done by software, it can be very slow.

*level* **v**

Toggle verbose mode, for use when single-stepping with **S**, **s**, or **m**. If *level* is omitted, then just the current source file and/or

subroutine name is printed when either changes. If *level* is 1 or greater, each C source line is printed before it is executed; if *level* is 2 or greater, each assembler statement is also printed. A **v** turns verbose mode off if it is on for any level.

**k** Kill the program being debugged.

`procedure(arg1,arg2,...)`  
`procedure(arg1,arg2,...)/m`

Execute the named procedure with the given arguments. Arguments can be integer, character, or string constants or names of variables accessible from the current procedure. The second form causes the value returned by the procedure to be printed according to format *m*. If no format is given, it defaults to **d**. This facility is only available if the program was loaded with the **-g** option.

#### *linenumber b commands*

Set a breakpoint at the given line. If a procedure name without a line number is given (for example, "proc:"), a breakpoint is placed at the first line in the procedure even if it was not compiled with the **-g** option. If no *linenumber* is given, a breakpoint is placed at the current line. If no *commands* are given, execution stops just before the breakpoint and control is returned to *sdb*. Otherwise the *commands* are executed when the breakpoint is encountered and execution continues. Multiple commands are specified by separating them with semicolons. If **k** is used as a command to execute at a breakpoint, control returns to *sdb*, instead of continuing execution.

**B** Print a list of the currently active breakpoints.

#### *linenumber d*

Delete a breakpoint at the given line. If no *linenumber* is given, then the breakpoints are deleted interactively. Each breakpoint location is printed and a line is read from the standard input. If the line begins with a **y** or **d**, then the breakpoint is deleted.

**D** Delete all breakpoints.

**l** Print the last executed line.

#### *linenumber a*

Announce. If *linenumber* is of the form *proc:number*, the command effectively does a *linenumber b l*. If *linenumber* is of the form *proc:*, the command effectively does a *proc: b T*.



Miscellaneous commands:

**!command**

The command is interpreted by *sh*(C).

**new-line**

If the previous command printed a source line, then advance the current line by one line and print the new current line. If the previous command displayed a memory location, then display the next memory location.

**end-of-file character**

Scroll. Print the next 10 lines of instructions, source or data depending on which was printed last. The end-of-file character is usually Control-D.

**<filename**

Read commands from *filename* until the end of file is reached, and then continue to accept commands from standard input. When *sdb* is told to display a variable by a command in such a file, the variable name is displayed along with the value. This command may not be nested; < may not appear as a command in a file.

**M** Print the address maps.

**M [?/] [\*] b e f**

Record new values for the address map. The arguments ? and / specify the text and data maps, respectively. The first segment (*b1*, *e1*, *f1*) is changed unless \* is specified; in which case, the second segment (*b2*, *e2*, *f2*) of the mapping is changed. If fewer than three values are given, the remaining map parameters are left unchanged.

**" string**

Print the given string. The C escape sequences of the form *\character* are recognized, where *character* is a nonnumeric character.

**q** Exit the debugger.

The following commands also exist and are intended only for debugging the debugger:

**V** Print the version number.

**Q** Print a list of procedures and files being debugged.

**Y** Toggle debug output.



## Files

---

a.out  
core

## See Also

---

cc(CP), a.out(F), core(F), syms(F).

sh(C) in the *User's Reference*.

## Warnings

---

When *sdb* prints the value of an external variable for which there is no debugging information, a warning is printed before the value. The size is assumed to be **int** (integer).

Data which are stored in text sections are indistinguishable from functions.

Line number information in optimized functions is unreliable, and some information may be missing.

## Notes

---

*sdb* operates transparently on either COFF or x.out format binary executables.

If a procedure is called when the program is *not* stopped at a breakpoint (such as when a core image is being debugged), all variables are initialized before the procedure is started. This makes it impossible to use a procedure which formats data from a core image.

## Standards Conformance

---

*sdb* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## size

---

print section sizes in bytes of common object files

### Syntax

---

**size** [-n] [-f] [-o] [-x] [-V] files

### Description

---

The *size* command produces section size information in bytes for each loaded section in the common object files. The size of the text, data, and bss (uninitialized data) sections is printed, as well as the sum of the sizes of these sections. If an archive file is input to the *size* command, the information for all archive members is displayed.

The **-n** option includes NOLOAD sections in the size.

The **-f** option produces full output, that is, it prints the size of every loaded section, followed by the section name in parentheses.

Numbers will be printed in decimal unless either the **-o** or the **-x** option is used, in which case they will be printed in octal or in hexadecimal, respectively.

The **-V** flag will supply the version information on the *size* command.

### See Also

---

**as**(CP), **cc**(CP), **ld**(CP), **a.out**(F), **ar**(F)

### Diagnostics

---

size: name: cannot open  
if *name* cannot be read.

size: name: bad magic  
if *name* is not an appropriate common object file.

## Notes

---

If you are using XENIX binaries, please refer to the manual entry for this utility in the *XENIX Development Guide* for information on the appropriate usage with XENIX binaries.

Since the size of bss sections is not known until link-edit time, the *size* command will not give the true total size of pre-linked objects.

## Standards Conformance

---

*size* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



## strip

---

strip symbol and line number information from a common object file

### Syntax

---

**strip** [-l] [-x] [-b] [-r] [-V] filename ...

### Description

---

The *strip* command strips the symbol table and line number information from common object files, including archives. Once this has been done, no symbolic debugging access will be available for that file; therefore, this command is normally run only on production modules that have been debugged and tested.

The amount of information stripped from the symbol table can be controlled by using any of the following options:

- l** Strip line number information only; do not strip any symbol table information.
- x** Do not strip static or external symbol information.
- b** Same as the **-x** option, but also do not strip scoping information (e.g., beginning and end of block delimiters).
- r** Do not strip static or external symbol information, or relocation information.
- V** Print the version of the strip command executing on the standard error output.

If there are any relocation entries in the object file and any symbol table information is to be stripped, *strip* will complain and terminate without stripping *filename* unless the **-r** option is used.

If the *strip* command is executed on a common archive file (see *ar(F)*) the archive symbol table will be removed. The archive symbol table must be restored by executing the *ar(CP)* command with the **s** option before the archive can be link-edited by the *ld(CP)* command. *strip* will produce appropriate warning messages when this situation arises.

The *strip* command is used to reduce the file storage overhead taken by the object file.

## Notes

---

If you are using XENIX binaries, please refer to the manual entry for this utility in the *XENIX Development Guide* for information on the appropriate usage with XENIX binaries.

## Files

---

*TMPDIR*/strip\*

temporary files

*TMPDIR* is usually */usr/tmp* but can be redefined by setting the environment variable **TMPDIR** (see *tmpnam()* in *tmpnam(S)*).

## See Also

---

ar(CP), as(CP), cc(CP), ld(CP), tmpnam(S), a.out(F), ar(F)

## Diagnostics

---

strip: name: cannot open

if *name* cannot be read.

strip: name: bad magic

if *name* is not an appropriate common object file.

strip: name: relocation entries present; cannot strip

if *name* contains relocation entries and the **-r** flag is not used, the symbol table information cannot be stripped.

## Standards Conformance

---

*strip* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## **tsort**

---

topological sort

### **Syntax**

---

`tsort [file]`

### **Description**

---

The *tsort* command produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

### **See Also**

---

`lorder`(CP).

### **Diagnostics**

---

Odd data: there is an odd number of fields in the input file.

### **Standards Conformance**

---

*tsort* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



## unget

---

undo a previous get of an SCCS file

### Syntax

---

**unget** [-rSID] [-s] [-n] files

### Description

---

The *unget* command undoes the effect of a **get -e** done prior to creating the intended new delta. If a directory is named, *unget* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of - is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

Keyletter arguments apply independently to each named file.

- |              |                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-rSID</b> | Uniquely identifies which delta is no longer intended. (This would have been specified by <i>get</i> as the "new delta.") The use of this keyletter is necessary only if two or more outstanding <i>gets</i> for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified <i>SID</i> is ambiguous, or if it is necessary and omitted on the command line. |
| <b>-s</b>    | Suppresses the printout, on the standard output, of the intended delta's <i>SID</i> .                                                                                                                                                                                                                                                                                                                           |
| <b>-n</b>    | Causes the retention of the gotten file which would normally be removed from the current directory.                                                                                                                                                                                                                                                                                                             |

### See Also

---

delta(C), get(CP), sact(CP)

### Standards Conformance

---

*unget* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## val

---

validate SCCS file

### Syntax

---

**val** -  
**val** [-s] [-rSID] [-mname] [-ytype] files

### Description

---

The *val* command determines if the specified *file* is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to *val* may appear in any order. The arguments consist of keyletter arguments, which begin with a -, and named files.

The *val* command has a special argument, -, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

The *val* command generates diagnostic messages on the standard output for each command line and file processed, and also returns a single 8-bit code upon exit as described below.

The keyletter arguments are defined as follows. The effects of any keyletter argument apply independently to each named file on the command line.

- |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-s</b>     | The presence of this argument silences the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line.                                                                                                                                                                                                                                                                                                       |
| <b>-rSID</b>  | The argument value <i>SID</i> (SCCS identification string) is an SCCS delta number. A check is made to determine if the <i>SID</i> is ambiguous (for example, <b>-r1</b> is ambiguous because it physically does not exist but implies 1.1, 1.2, etc., which may exist) or invalid (for example, <b>-r1.0</b> or <b>-r1.1.0</b> are invalid because neither case can exist as a valid delta number). If the <i>SID</i> is valid and not ambiguous, a check is made to determine if it actually exists. |
| <b>-mname</b> | The argument value <i>name</i> is compared with the SCCS %M% keyword in <i>file</i> .                                                                                                                                                                                                                                                                                                                                                                                                                  |

**-ytype**      The argument value *type* is compared with the SCCS %Y% keyword in *file*.

The 8-bit code returned by *val* is a disjunction of the possible errors, that is, it can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

- bit 0 = missing file argument;
- bit 1 = unknown or duplicate keyletter argument;
- bit 2 = corrupted SCCS file;
- bit 3 = cannot open file or file not SCCS;
- bit 4 = *SID* is invalid or ambiguous;
- bit 5 = *SID* does not exist;
- bit 6 = %Y%, -y mismatch;
- bit 7 = %M%, -m mismatch;

Note that *val* can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned - a logical **OR** of the codes generated for each command line and file processed.

## See Also

---

admin(CP), delta(CP), get(CP), prs(CP), help(CP)

## Diagnostics

---

Use *help*(CP) for explanations.

## Notes

---

The *val* command can process up to 50 files on a single command line. Any number above 50 will produce a **core** dump.

## Standards Conformance

---

*val* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



## VC

### version control

---

## Syntax

---

`vc [-a] [-t] [-cchar] [-s] [keyword=value ... keyword=value]`

## Description

---

The `vc` command copies lines from the standard input to the standard output under control of its *arguments* and *control statements* encountered in the standard input. In the process of performing the copy operation, user declared *keywords* may be replaced by their string *value* when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as `vc` command arguments.

A control statement is a single line beginning with a control character, except as modified by the `-t` keyletter (see below). The default control character is colon (:), except as modified by the `-c` keyletter (see below). Input lines beginning with a backslash (\) followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A keyword is composed of 9 or fewer alphanumerics; the first must be alphabetic. A value is any ASCII string that can be created with `ed(C)`; a numeric value is an unsigned string of digits. Keyword values may not contain blanks or tabs.

Replacement of keywords by values is done whenever a keyword surrounded by control characters is encountered on a version control statement. The `-a` keyletter (see below) forces replacement of keywords in *all* lines of text. An uninterpreted control character may be included in a value by preceding it with \. If a literal \ is desired, then it too must be preceded by \.

### Keyletter Arguments

- a** Forces replacement of keywords surrounded by control characters with their assigned value in *all* text lines and not just in `vc` statements.

- t All characters from the beginning of a line up to and including the first *tab* character are ignored for the purpose of detecting a control statement. If one is found, all characters up to and including the *tab* are discarded.
- cchar Specifies a control character to be used in place of :.
- s Silences warning messages (not error) that are normally printed on the diagnostic output.

### Version Control Statements

:dcl keyword[, ..., keyword]

Used to declare keywords. All keywords must be declared.

:asg keyword=value

Used to assign values to keywords. An **asg** statement overrides the assignment for the corresponding keyword on the **vc** command line and all previous **asg**'s for that keyword. Keywords declared, but not assigned values have null values.

:if condition

:

:end

Used to skip lines of the standard input. If the condition is true, all lines between the *if* statement and the matching *end* statement are copied to the standard output. If the condition is false, all intervening lines are discarded, including control statements. Note that intervening *if* statements and matching *end* statements are recognized solely for the purpose of maintaining the proper *if-end* matching.

The syntax of a condition is:

|         |                                                 |
|---------|-------------------------------------------------|
| <cond>  | ::= [ "not" ] <or>                              |
| <or>    | ::= <and>   <and> "   " <or>                    |
| <and>   | ::= <exp>   <exp> "&" <and>                     |
| <exp>   | ::= "(" <or> ")"   <value> <op> <value>         |
| <op>    | ::= "="   "!="   "<"   ">"                      |
| <value> | ::= <arbitrary ASCII string>   <numeric string> |

The available operators and their meanings are:

|     |                                                                                                              |
|-----|--------------------------------------------------------------------------------------------------------------|
| =   | equal                                                                                                        |
| !=  | not equal                                                                                                    |
| &   | and                                                                                                          |
|     | or                                                                                                           |
| >   | greater than                                                                                                 |
| <   | less than                                                                                                    |
| ()  | used for logical groupings                                                                                   |
| not | may only occur immediately after the <i>if</i> , and when present, inverts the value of the entire condition |

The > and < operate only on unsigned integer values (for example, : 012 > 12 is false). All other operators take strings as arguments (for example, : 012 != 12 is true). The precedence of the operators (from highest to lowest) is:

|   |    |   |   |                         |
|---|----|---|---|-------------------------|
| = | != | > | < | all of equal precedence |
| & |    |   |   |                         |
|   |    |   |   |                         |

Parentheses may be used to alter the order of precedence.

Values must be separated from operators or parentheses by at least one blank or tab.

**::text** Used for keyword replacement on lines that are copied to the standard output. The two leading control characters are removed, and keywords surrounded by control characters in text are replaced by their value before the line is copied to the output file. This action is independent of the **-a** keyletter.

**:on**

**:off** Turn on or off keyword replacement on all lines.

**:ctl char**

Change the control character to char.

**:msg message**

Prints the given message on the diagnostic output.

**:err message**

Prints the given message followed by:

**ERROR:** err statement on line ... (915)

on the diagnostic output. vc halts execution and returns an exit code of 1.



## **See Also**

---

ed(C)

## **Exit Codes**

---

0 - normal  
1 - any error

# what

---

identify SCCS files

## Syntax

---

**what** [-s] files

## Description

---

The *what* command searches the given files for all occurrences of the pattern that *get*(CP) substitutes for %Z% (this is @(#) at this printing) and prints out what follows until the first ``, >, new-line, \,` or null character. For example, if the C program in file **f.c** contains

```
char ident[] = "@(#)identification information";
```

and **f.c** is compiled to yield **f.o** and **a.out**, then the command

```
what f.c f.o a.out
```

will print

```
f.c: identification information
```

```
f.o: identification information
```

```
a.out: identification information
```

The *what* command is intended to be used in conjunction with the command *get*(CP), which automatically inserts identifying information, but it can also be used where the information is inserted manually. Only one option exists:

**-s**           Quit after finding the first occurrence of pattern in each file.

## See Also

---

*get*(CP).

## Diagnostics

---

Exit status is 0 if any matches are found, otherwise 1.

**Notes**

---

It is possible that an unintended occurrence of the pattern @(#) could be found just by chance, but this causes no harm in nearly all cases.

**Standards Conformance**

---

*what* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



# x286emul

---

emulate XENIX 80286

## Syntax

---

**x286emul** [ arg ... ] **prog286**

## Description

---

*x286emul* is an emulator that allows programs from XENIX System V/286 Release 2.3 or XENIX System V/286 Release 2.3.2 on the Intel 80286 to run on the Intel 80386 processor under System V Release 3.2.

The system recognizes an attempt to *exec*(S) a 286 program, and automatically *exec*'s the 286 emulator with the 286 program name as an additional argument. It is not necessary to specify the *x286emul* emulator on the command line. The 286 programs can be invoked using the same command format as on the XENIX System V/286.

*x286emul* reads the 286 program's text and data into memory and maps them through the LDT [via *sysi86*(S)] as 286 text and data segments. It also fills in the jam area, which is used by XENIX programs to do system calls and signal returns. *x286emul* starts the 286 program by jumping to its entry point.

When the 286 program attempts to do a system call, *x286emul* takes control. It does any conversions needed between the 286 system call and the equivalent 386 system call, and performs the 386 system call. The results are converted to the form the 286 program expects, and the 286 program is resumed.

The following are some of the differences between a program running on a 286 and a 286 program using *x286emul* on a 386:

Attempts to unlink or write on the 286 program will fail on the 286 with ETXTBSY. Under *x286emul*, they will not fail.

*ptrace*(S) is not supported under *x286emul*.

The 286 program must be readable for the emulator to read it.

## Files

---

/bin/x286emul

The emulator must have this name and be in **/bin** if it is to be automatically invoked when *exec*(S) is used on a 286 program.

## xref

---

cross-references C programs

### Syntax

---

`xref [ file ... ]`

### Description

---

*xref* reads the named *files* or the standard input if no file is specified and prints a cross reference consisting of lines of the form

|            |          |                  |
|------------|----------|------------------|
| identifier | filename | line numbers ... |
|------------|----------|------------------|

Function definition is indicated by a plus sign (+) preceding the line number.

### See Also

---

`cref(CP)`



## xstr

extracts strings from C programs

### Syntax

**xstr** [-c] [-] [ file ]

### Description

*xstr* maintains a file *strings* into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, most useful if they are also read-only.

The command

**xstr -c name**

will extract the strings from the C source in *name*, replacing string references by expressions of the form (*&xstr[number]*) for some number. An appropriate declaration of *xstr* is prepended to the file. The resulting C text is placed in the file *x.c*, to then be compiled. The strings from this file are placed in the *strings* data base if they are not there already. Repeated strings and strings which are suffices of existing strings do not cause changes to the data base.

After all components of a large program have been compiled, a file *xs.c* declaring the common *xstr* space can be created by a command of the form

**xstr -c name1 name2 name3 ...**

This *xs.c* file should then be compiled and loaded with the rest of the program. If possible, the array can be made read-only (shared) saving space and swap overhead.

*xstr* can also be used on a single file. A command

**xstr name**

creates files *x.c* and *xs.c* as before, without using or affecting any *strings* file in the same directory.

It may be useful to run *xstr* after the C preprocessor if any macro definitions yield strings or if there is conditional code which contains strings which may not, in fact, be needed. *xstr* reads from its standard

input when the argument - is given. An appropriate command sequence for running *xstr* after the C preprocessor is:

```
cc -E name.c | xstr -c -
cc -c x.c
mv x.o name.o
```

*xstr* does not touch the file *strings* unless new items are added, thus *make* can avoid remaking *xs.o* unless truly necessary.

## Files

---

|          |                                                         |
|----------|---------------------------------------------------------|
| strings  | Data base of strings                                    |
| x.c      | Massaged C source                                       |
| xs.c     | C source for definition of array "xstr"                 |
| /tmp/xs* | Temp file when "xstr name" doesn't touch <i>strings</i> |

## See Also

---

mkstr(CP)

## Credit

---

This utility was developed at the University of California at Berkeley and is used with permission.

## Notes

---

If a string is a suffix of another string in the data base, but the shorter string is seen first by *xstr*, both strings will be placed in the data base when just placing the longer one there will do.

# yacc

---

yet another compiler-compiler

## Syntax

---

`yacc [ -vdlit ] grammar`

## Description

---

The *yacc* command converts a context-free grammar into a set of tables for a simple automaton which executes a parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, **y.tab.c**, must be compiled by the C compiler to produce a program *yyparse*. This program must be loaded with the lexical analyzer program, *yylex*, as well as *main* and *yyerror*, an error-handling routine. These routines must be supplied by the user; *lex*(CP) is useful for creating lexical analyzers usable by *yacc*.

If the **-v** flag is given, the file **y.output** is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the **-d** flag is used, the file **y.tab.h** is generated with the **#define** statements that associate the *yacc*-assigned "token codes" with the user-declared "token names". This allows source files other than **y.tab.c** to access the token codes.

If the **-l** flag is given, the code produced in **y.tab.c** will *not* contain any **#line** constructs. This should only be used after the grammar and the associated actions are fully debugged.

Runtime debugging code is always generated in **y.tab.c** under conditional compilation control. By default, this code is not included when **y.tab.c** is compiled. However, when *yacc*'s **-t** option is used, this debugging code will be compiled by default. Independent of whether the **-t** option was used, the runtime debugging code is under the control of **YYDEBUG**, a preprocessor symbol. If **YYDEBUG** has a non-zero value, then the debugging code is included. If its value is zero, then the code will not be included. The size and execution time of a program produced without the runtime debugging code will be smaller and slightly faster.



## Files

---

|                       |                                 |
|-----------------------|---------------------------------|
| y.output              |                                 |
| y.tab.c               |                                 |
| y.tab.h               | defines for token names         |
| yacc.tmp,             |                                 |
| yacc.debug, yacc.acts | temporary files                 |
| /usr/lib/yaccpar      | parser prototype for C programs |

## See Also

---

lex(CP)

## Diagnostics

---

The number of reduce-reduce and shift-reduce conflicts is reported on the standard error output; a more detailed report is found in the **y.output** file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

## Caveat

---

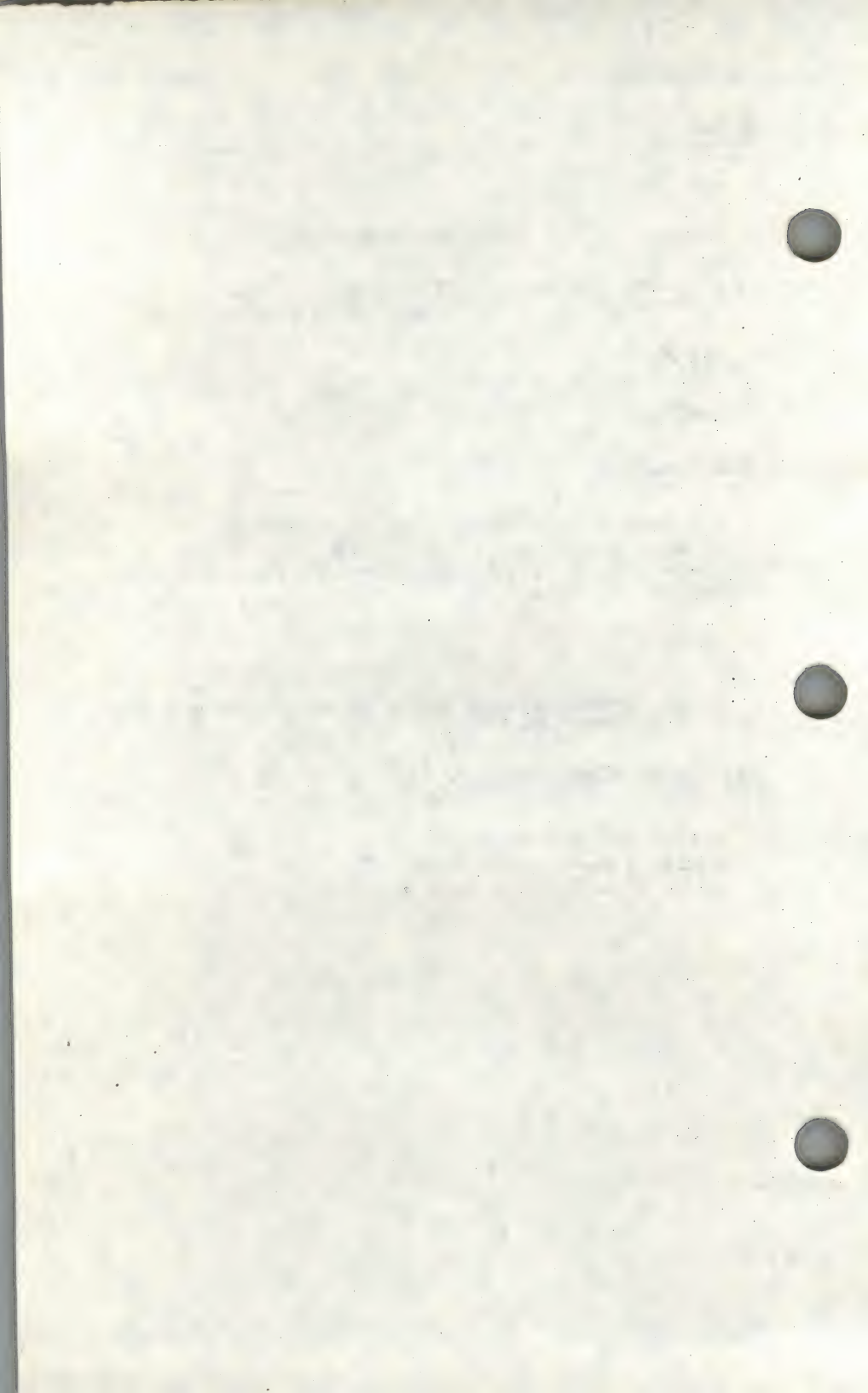
Because file names are fixed, at most one *yacc* process can be active in a given directory at a given time.

## Standards Conformance

---

*yacc* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



# Contents

---

## *System Services (S)*

|                                                                                                          |                                                                |
|----------------------------------------------------------------------------------------------------------|----------------------------------------------------------------|
| <b>Intro</b>                                                                                             | introduces system services, library routines and error numbers |
| <b>a64l, l64a</b>                                                                                        | convert between long integer and base-64 ASCII string          |
| <b>abort</b>                                                                                             | generate an abort fault                                        |
| <b>abs</b>                                                                                               | return integer absolute value                                  |
| <b>acceptable_password</b>                                                                               | determine if password is cryptic                               |
| <b>access</b>                                                                                            | determine accessibility of a file                              |
| <b>acct</b>                                                                                              | enable or disable process accounting                           |
| <b>alarm</b>                                                                                             | set a process alarm clock                                      |
| <b>assert</b>                                                                                            | verify program assertion                                       |
| <b>atexit</b>                                                                                            | calls a process at termination                                 |
| <b>atof, atoi, atol</b>                                                                                  | converts ASCII to numbers                                      |
| <b>audit_open,</b><br><b>audit_read,</b><br><b>audit_close</b>                                           | open and access audit session data on a record basis           |
| <b>authaudit</b>                                                                                         | produce audit records due to authentication events             |
| <b>authcap</b>                                                                                           | get information from the authentication database               |
| <b>bessel: j0, j1, jn,</b><br><b>y0, y1, yn</b>                                                          | bessel functions                                               |
| <b>brk, sbrk</b>                                                                                         | change data segment space allocation                           |
| <b>brkctl</b>                                                                                            | allocates data in a 286 far segment                            |
| <b>bsearch</b>                                                                                           | binary search a sorted table                                   |
| <b>cfspeed:</b><br><b>cfgetspeed,</b><br><b>cfgetospeed,</b><br><b>cfsetspeed,</b><br><b>cfsetospeed</b> | baud rate functions                                            |
| <b>chdir</b>                                                                                             | change working directory                                       |
| <b>check_basic_data_structures</b>                                                                       | verify machine is suitable for security port                   |
| <b>chmod</b>                                                                                             | change mode of file                                            |
| <b>chown</b>                                                                                             | change owner and group of a file                               |
| <b>chroot</b>                                                                                            | change root directory                                          |
| <b>chsize</b>                                                                                            | changes the size of a file.                                    |
| <b>clock</b>                                                                                             | report CPU time used                                           |



|                           |                                                      |
|---------------------------|------------------------------------------------------|
| <b>close</b>              | close a file descriptor                              |
| <b>conv: toupper,</b>     |                                                      |
| <b>tolower,</b>           |                                                      |
| <b>_toupper,</b>          |                                                      |
| <b>_tolower, toascii,</b> |                                                      |
| <b>todigit, toint</b>     | routines used to translate characters                |
| <b>creat</b>              | create a new file or rewrite an existing one         |
| <b>creatsem</b>           | creates an instance of a binary semaphore            |
| <b>crypt</b>              | password and file encryption functions               |
| <b>ctermid</b>            | generate file name for terminal                      |
| <b>ctime, localtime,</b>  |                                                      |
| <b>gmtime, asctime,</b>   |                                                      |
| <b>cftime, ascftime,</b>  |                                                      |
| <b>strftime, tzset</b>    | convert date and time to string                      |
| <b>ctype</b>              | character handling routines                          |
| <b>curses</b>             | terminal screen handling and optimization package    |
| <b>cuserid</b>            | get character login name of the user                 |
| <b>dblock</b>             | lock the entire Authentication database              |
| <b>dbm: dbminit,</b>      |                                                      |
| <b>fetch, store,</b>      |                                                      |
| <b>delete, firstkey,</b>  |                                                      |
| <b>nextkey</b>            | performs database functions                          |
| <b>defopen, defread</b>   | reads default entries                                |
| <b>dial</b>               | establish an outgoing terminal line connection       |
| <b>difftime</b>           | computes the difference between time values          |
| <b>directory:</b>         |                                                      |
| <b>opendir, readdir,</b>  |                                                      |
| <b>telldir, seekdir,</b>  |                                                      |
| <b>rewinddir,</b>         |                                                      |
| <b>closedir</b>           | directory operations                                 |
| <b>discr</b>              | check discretionary attributes of files and programs |
| <b>div</b>                | divides integers                                     |
| <b>drand48,</b>           |                                                      |
| <b>erand48, lrand48,</b>  |                                                      |
| <b>nrand48,</b>           |                                                      |
| <b>mrand48,</b>           |                                                      |
| <b>jrand48, srand48,</b>  |                                                      |
| <b>seed48, lcong48</b>    | generate uniformly distributed pseudo-random numbers |
| <b>dup</b>                | duplicate an open file descriptor                    |
| <b>dup2</b>               | duplicate an open file descriptor                    |
| <b>ecvt, fcvt, gcvt</b>   | convert floating-point number to string              |
| <b>end, etext, edata</b>  | last locations in program                            |
| <b>erf, erfc</b>          | error function and complementary error function      |

|                            |                                                                         |
|----------------------------|-------------------------------------------------------------------------|
| <b>ev_block</b>            | wait until the queue contains an event                                  |
| <b>ev_close</b>            | close the event queue and all associated devices                        |
| <b>ev_count</b>            | returns the number of events currently in the queue                     |
| <b>ev_flush</b>            | discard all events currently in the queue                               |
| <b>ev_getdev</b>           | gets a list of devices feeding an event queue                           |
| <b>ev_getemask</b>         | return the current event mask                                           |
| <b>ev_gindev</b>           | include/exclude devices for event input                                 |
| <b>ev_init</b>             | invokes the event manager                                               |
| <b>ev_pop</b>              | pop the next event off the queue                                        |
| <b>ev_read</b>             | read the next event in the queue                                        |
| <b>ev_resume</b>           | restart a suspended queue                                               |
| <b>ev_setemask</b>         | sets event mask                                                         |
| <b>ev_suspend</b>          | suspends an event queue.                                                |
| <b>exec: execl, execv,</b> |                                                                         |
| <b>execle, execve,</b>     |                                                                         |
| <b>execlp, execvp</b>      | execute a file                                                          |
| <b>execseg</b>             | makes a data region executable.                                         |
| <b>exit, _exit</b>         | terminate process                                                       |
| <b>exp, log, log10,</b>    |                                                                         |
| <b>pow, sqrt</b>           | exponential, logarithm, power, square root functions                    |
| <b>fclose, fflush</b>      | close or flush a stream                                                 |
| <b>fcntl</b>               | file control                                                            |
| <b>ferror, feof,</b>       |                                                                         |
| <b>clearerr, fileno</b>    | stream status inquiries                                                 |
| <b>fgetpos</b>             | gets and stores the current value of a stream's file position indicator |
| <b>field</b>               | FIELD library routines                                                  |
| <b>fields</b>              | return status based on fields of authentication database                |
| <b>fieldtype</b>           | FIELDTYPE library routines                                              |
| <b>floor, ceil, fmod,</b>  |                                                                         |
| <b>fabs</b>                | floor, ceiling, remainder, absolute value functions                     |
| <b>fopen, freopen,</b>     |                                                                         |
| <b>fdopen</b>              | open a stream                                                           |
| <b>fork</b>                | create a new process                                                    |
| <b>form</b>                | FORM library routines                                                   |
| <b>fpgetround,</b>         |                                                                         |
| <b>fpsetround,</b>         |                                                                         |
| <b>fpgetmask,</b>          |                                                                         |
| <b>fpsetmask,</b>          |                                                                         |
| <b>fpgetsticky,</b>        |                                                                         |
| <b>fpsetsticky</b>         | IEEE floating point environment control                                 |
| <b>fread, fwrite</b>       | binary input/output                                                     |
| <b>frexp, ldexp,</b>       |                                                                         |
| <b>modf</b>                | manipulate parts of floating-point numbers                              |

**fseek, rewind,**  
**ftell**  
**fsetpos**  
**ftw**  
**gamma**  
**getc, getchar,**  
**fgetc, getw**  
**getcwd**  
**getdents**

**getdvagent,**  
**getdvagname,**  
**setdvagent,**  
**enddvagent,**  
**putdvagname,**  
**copydvagent**  
**getenv**  
**getgrent,**  
**getgrgid,**  
**getgrnam,**  
**setgrent,**  
**endgrent,**  
**fgetgrent**  
**getgroups**  
**gethz**

**getlogin**  
**getluid**  
**getmsg**  
**getopt**  
**getpass**  
**getpasswd**  
**getpid, getpgrp,**  
**getppid**  
**getprcmnt,**  
**getprcmnam,**  
**setprcmnt,**  
**endprcmnt,**  
**putprcmnam**  
**getprdfent,**  
**getprdfnam,**  
**setprdfent,**  
**endprdfent,**  
**putprdfnam**

reposition a file pointer in a stream  
sets the file position indicator for a stream  
walk a file tree  
log gamma function

get character or word from a stream  
get path name of current working directory  
read directory entries and put in a file-system-independent format

manipulate device assignment database entry  
return value for environment name

get group file entry  
get supplementary group ID's  
return the frequency of the system clock in ticks per second

get login name  
get login user ID  
get next message off a stream  
get option letter from argument vector  
read a password  
read or clear a password

get process, process group, and parent process IDs

manipulate command control database entry

manipulate default control database entry



getprfient,  
 getprfinam,  
 setprfient,  
 endprfient,  
 putprfinam  
 getpriv  
 getprpwent,  
 getprpwuid,  
 getprpwnam,  
 setprpwent,  
 endprpwent,  
 putprpwnam  
 getprtcent,  
 getprtcnam,  
 setprtcent,  
 endprtcent,  
 putprtcnam  
 getpw  
 getpwent,  
 getpwuid,  
 getpwnam,  
 setpwent,  
 endpwent,  
 fgetpwent  
 gets, fgets  
 getuid, geteuid,  
 getgid, getegid

manipulate file control database entry  
 get system privileges associated with this process

manipulate protected password database entry

manipulate terminal control database entry  
 get name from UID

get password file entry  
 get a string from a stream

get real user, effective user, real group, and effective group IDs

getut: getutent,  
 getutid, getutline,  
 pututline,  
 setutent,  
 endutent,  
 utmpname  
 hsearch, hcreate,  
 hdestroy  
 hypot  
 identity  
 ioctl  
 isnan: isnand,  
 isnanf  
 item  
 kill  
 l3tol, ltol3

access utmp file entry

manage hash search tables  
 euclidean distance function  
 get or check uids or gids from program start  
 control device

test for floating point NaN (Not-A-Number)  
 CRT item routines  
 send a signal to a process or a group of processes  
 convert between 3-byte integers and long integers

|                                                |                                                                   |
|------------------------------------------------|-------------------------------------------------------------------|
| <b>labs</b>                                    | converts to absolute value                                        |
| <b>ldahread</b>                                | read the archive header of a member of an archive file            |
| <b>ldclose, ldaclose</b>                       | close a common object file                                        |
| <b>ldfhread</b>                                | read the file header of a common object file                      |
| <b>ldgetname</b>                               | retrieve symbol name for common object file symbol table entry    |
| <b>ldiv</b>                                    | divides long integers                                             |
| <b>ldlread, ldlnit, lditem</b>                 | manipulate line number entries of a common object file function   |
| <b>ldlseek, ldlnseek</b>                       | seek to line number entries of a section of a common object file  |
| <b>ldohseek</b>                                | seek to the optional file header of a common object file          |
| <b>ldopen, ldaopen</b>                         | open a common object file for reading                             |
| <b>ldrseek, ldnrseek</b>                       | seek to relocation entries of a section of a common object file   |
| <b>ldshread, ldnshread</b>                     | read an indexed/named section header of a common object file      |
| <b>ldsseek, ldnsseek</b>                       | seek to an indexed/named section of a common object file          |
| <b>ldtbindex</b>                               | compute the index of a symbol table entry of a common object file |
| <b>ldtbread</b>                                | read an indexed symbol table entry of a common object file        |
| <b>ldtbseek</b>                                | seek to the symbol table of a common object file                  |
| <b>libwindows</b>                              | windowing terminal function library                               |
| <b>link</b>                                    | link to a file                                                    |
| <b>lock</b>                                    | locks a process in primary memory                                 |
| <b>lockf</b>                                   | record locking on files                                           |
| <b>locking</b>                                 | locks or unlocks a file region for reading or writing             |
| <b>logname</b>                                 | return login name of user                                         |
| <b>lsearch, lfind</b>                          | linear search and update                                          |
| <b>lseek</b>                                   | move read/write file pointer                                      |
| <b>malloc, free, realloc, calloc</b>           | allocates main memory                                             |
| <b>matherr</b>                                 | error-handling function                                           |
| <b>memmove</b>                                 | copies characters between objects                                 |
| <b>memory:</b>                                 |                                                                   |
| <b>memccpy, memchr, memcmp, memcpy, memset</b> | memory operations                                                 |

|                       |                                                             |
|-----------------------|-------------------------------------------------------------|
| <b>menu</b>           | CRT menu routines                                           |
| <b>mkdir</b>          | make a directory                                            |
| <b>mkfifo</b>         | make a FIFO special file                                    |
| <b>mknod</b>          | make a directory or a special or ordinary file or a FIFO    |
| <b>mktemp</b>         | make a unique file name                                     |
| <b>mktime</b>         | converts local time to calendar time                        |
| <b>monitor</b>        | prepare execution profile                                   |
| <b>mount</b>          | mount a file system                                         |
| <b>msgctl</b>         | message control operations                                  |
| <b>msgget</b>         | get message queue                                           |
| <b>msgop: msgsnd,</b> |                                                             |
| <b>msgrcv</b>         | message operations                                          |
| <b>nap</b>            | suspends execution for a short interval                     |
| <b>nice</b>           | change priority of a process                                |
| <b>nl_ascxtime,</b>   |                                                             |
| <b>nl_cxtime</b>      | format date and time                                        |
| <b>nl_init</b>        | initializes native language support operation               |
| <b>nl_langinfo</b>    | language information                                        |
| <b>nl_printf,</b>     |                                                             |
| <b>nl_fprintf,</b>    |                                                             |
| <b>nl_sprintf</b>     | formats native language output                              |
| <b>nl_scanf,</b>      |                                                             |
| <b>nl_fscanf,</b>     |                                                             |
| <b>nl_sscanf</b>      | converts formatted native language input                    |
| <b>nl_strcmp,</b>     |                                                             |
| <b>nl_strncmp</b>     | compare native language strings                             |
| <b>nlist</b>          | get entries from name list                                  |
| <b>open</b>           | open for reading or writing                                 |
| <b>opensem</b>        | opens a semaphore                                           |
| <b>paccess</b>        | used in conjunction with ptrace for tracing a child process |
| <b>panel</b>          | PANEL library routines                                      |
| <b>passlen</b>        | determine minimum password length                           |
| <b>pathconf</b>       | get configurable pathname variables                         |
| <b>pause</b>          | suspend process until signal                                |
| <b>perror, errno,</b> |                                                             |
| <b>sys_errlist,</b>   |                                                             |
| <b>sys_nerr</b>       | system error messages                                       |
| <b>pipe</b>           | create an interprocess channel                              |
| <b>plock</b>          | lock process, text, or data in memory                       |
| <b>plot</b>           | graphics interface subroutines                              |
| <b>poll</b>           | STREAMS input/output multiplexing                           |
| <b>popen, pclose</b>  | initiate pipe to/from a process                             |



|                         |                                                |
|-------------------------|------------------------------------------------|
| <b>printf, fprintf,</b> | print formatted output                         |
| <b>sprintf</b>          | controls active processes or process groups    |
| <b>proct</b>            | execution time profile                         |
| <b>profil</b>           | process trace                                  |
| <b>ptrace</b>           |                                                |
| <b>putc, putchar,</b>   | put character or word on a stream              |
| <b>fputc, putw</b>      | change or add value to environment             |
| <b>putenv</b>           | send a message on a stream                     |
| <b>putmsg</b>           | write password file entry                      |
| <b>putpwent</b>         | put a string on a stream                       |
| <b>puts, fputs</b>      |                                                |
| <b>pw_mapping:</b>      |                                                |
| <b>pw_nametoid,</b>     |                                                |
| <b>pw_idtoname,</b>     |                                                |
| <b>gr_nametoid,</b>     |                                                |
| <b>gr_idtoname</b>      | map between user and group names and IDs       |
| <b>qsort</b>            | quicker sort                                   |
| <b>raise</b>            | send signal <i>sig</i> to execution program    |
| <b>rand, srand</b>      | simple random-number generator                 |
| <b>randomword</b>       | generate a pronounceable password              |
| <b>rdchk</b>            | checks to see if there is data to be read      |
| <b>read</b>             | read from file                                 |
| <b>regcmp, regex</b>    | compile and execute regular expression         |
| <b>regexp</b>           | regular expression compile and match routines  |
| <b>remove</b>           | removes filename                               |
| <b>rename</b>           | changes filename                               |
| <b>rmdir</b>            | remove a directory                             |
| <b>scanf, fscanf,</b>   |                                                |
| <b>sscanf</b>           | convert formatted input                        |
| <b>sdenter, sdleave</b> | synchronizes access to a shared data segment   |
| <b>sdget, sdfree</b>    | attaches and detaches a shared data segment.   |
| <b>sdgetv, sdwaitv</b>  | synchronizes shared data access                |
| <b>seed: getseed,</b>   |                                                |
| <b>setseed</b>          | obtain or set seed for random number generator |
| <b>select</b>           | synchronous I/O multiplexing                   |
| <b>semctl</b>           | semaphore control operations                   |
| <b>semget</b>           | get set of semaphores                          |
| <b>semop</b>            | semaphore operations                           |
| <b>setbuf, setvbuf</b>  | assign buffering to a stream                   |
| <b>setgroups</b>        | set group access list                          |
| <b>setjmp, longjmp</b>  | non-local goto                                 |
| <b>setlocale</b>        | set or read international environment          |
| <b>setluid</b>          | set login user ID                              |
| <b>setpgid</b>          | set process group ID for job control           |

**setpgrp**  
**setpriv**  
**setsid**  
**setuid, setgid**

**shmctl**  
**shmget**  
**shmop: shmat,**

**shmdt**  
**shutdn**

**sigaction**

**signal**

**sigpending**

**sigprocmask**

**sigsem**

**sigset**

**sigsetjmp,**

**siglongjmp**

**sigsuspend**

**sinh, cosh, tanh**

**sleep**

**sputl, sgetl**

**ssignal, gsignal**

**stat, fstat**

**statfs, fstatfs**

**stdio**

**stdipc: ftok**

**stime**

**stopio**

**strerror**

**strftime**

**string: strcat,**

**strdup, strncat,**

**strcmp, strncmp,**

**strcpy, strncpy,**

**strlen, strchr,**

**strrchr, strpbrk,**

**strspn, strcspn,**

**strtok**

**strtod, atof**

**strtol, atol, atoi**

**strxfrm,**

**strnxfrm, strcoll,**

**strncoll**

set process group ID

set system privileges for this process

create session and set process ID

set user and group IDs

shared memory control operations

get shared memory segment identifier

shared memory operations

flushes block I/O and halts the CPU

examine and change signal action

specify what to do upon receipt of a signal

examine pending signals

examine and change blocked signals.

signals a process waiting on a semaphore

manipulate signal sets

non-local jumps

wait for signal

hyperbolic functions

suspend execution for interval

access long integer data in a machine-independent fashion

software signals

get file status

get file system information

standard buffered input/output package

standard interprocess communication package

set time

stop further I/O to an open file

gets error message pointer from last routine call error

format date/time string

string operations

convert string to double-precision number

convert string to integer

handles collation of strings

|                             |                                                    |
|-----------------------------|----------------------------------------------------|
| <b>subsystems</b>           | manipulation routines for Subsystems database      |
| <b>swab</b>                 | swap bytes                                         |
| <b>sync</b>                 | update super block                                 |
| <b>sysconf</b>              | get configurable system variables                  |
| <b>sysfs</b>                | get file system type information                   |
| <b>sysi86</b>               | machine-specific functions                         |
| <b>system</b>               | issue a shell command                              |
| <b>tam</b>                  | TAM transition libraries                           |
| <b>tcdrain, tcflow,</b>     |                                                    |
| <b>tcflush,</b>             |                                                    |
| <b>tcsendbreak</b>          | line control functions                             |
| <b>tcgetattr,</b>           |                                                    |
| <b>tcsetattr</b>            | state functions                                    |
| <b>tcgetpgrp,</b>           |                                                    |
| <b>tcsetpgrp</b>            | process group id functions                         |
| <b>tgetent, tgetnum,</b>    |                                                    |
| <b>tgetflag, tgetstr,</b>   |                                                    |
| <b>tgoto, tputs</b>         | performs terminal functions                        |
| <b>terminfo</b>             | terminal description database.                     |
| <b>time</b>                 | get time                                           |
| <b>times</b>                | get process and child process times                |
| <b>tmpfile</b>              | create a temporary file                            |
| <b>tmpnam,</b>              |                                                    |
| <b>tempnam</b>              | create a name for a temporary file                 |
| <b>trig: sin, cos, tan,</b> |                                                    |
| <b>asin, acos, atan,</b>    |                                                    |
| <b>atan2</b>                | trigonometric functions                            |
| <b>tsearch, tfind,</b>      |                                                    |
| <b>tdelete, twalk</b>       | manage binary search trees                         |
| <b>ttyname, isatty</b>      | find name of a terminal                            |
| <b>ttyslot</b>              | find the slot in the utmp file of the current user |
| <b>uadmin</b>               | administrative control                             |
| <b>ulimit</b>               | get and set user limits                            |
| <b>umask</b>                | set and get file creation mask                     |
| <b>umount</b>               | unmount a file system                              |
| <b>uname</b>                | get name of current system                         |
| <b>ungetc</b>               | push character back into input stream              |
| <b>unlink</b>               | remove directory entry                             |
| <b>ustat</b>                | get file system statistics                         |
| <b>utime</b>                | set file access and modification times             |
| <b>varargs</b>              | variable argument list                             |
| <b>vprintf, vsprintf,</b>   |                                                    |
| <b>vsprintf</b>             | print formatted output of a varargs argument list  |
| <b>wait, waitpid</b>        | wait for child process to stop or terminate        |



**waitsem,  
nbwaitsem**

awaits and checks access to a resource governed by a semaphore

**write**

write on a file

**xlist, fxlist**

gets name list entries from files



## Intro

introduces system services, library routines and error numbers

## Syntax

```
#include <errno.h>
```

## Description

This section describes all system services. System services include all routines or system calls that are available in the operating system kernel. These routines are available to a C program automatically as part of the standard library **libc**. Other routines are available in a variety of libraries. On 386 systems, Small, Middle, and Large programs for 286 processes and Small model programs for 386 processes are provided.

To use routines in a program that are not part of the standard library **libc**, the appropriate library must be linked. This is done by specifying **-l name** to the compiler or linker, where *name* is the name listed below. For example **-l m**, and **-l termcap** are specifications to the linker to search the named libraries for routines to be linked to the object module. The names of the available libraries are:

- c**        The standard library containing all system call interfaces, Standard I/O routines, and other general purpose services.
- m**        The standard math library.
- termcap** Routines for accessing the *termcap* data base describing terminal characteristics.
- curses**   Screen and cursor manipulation routines.
- dbm**      Data base management routines.
- event**    Event hardware API routines.
- x**        The standard UNIX library.

Most services that are part of the operating system kernel have one or more error returns. An error condition is indicated by an otherwise impossible returned value. This is almost always -1; the individual descriptions specify the details. An error number is also made available in the external variable *errno*. *errno* is not cleared on successful calls, so it should be tested only after an error has been indicated.



All of the possible error numbers are not listed in each system call description because many errors are possible for most of the calls. The following is a complete list of the error numbers and their names as defined in `<errno.h>`.

1 EPERM Not owner:

Typically, this error indicates an attempt to modify a file in some way forbidden except to its owner or super-user. It is also returned for attempts by ordinary users to do things allowed only to the super-user.

2 ENOENT No such file or directory:

This error occurs when a filename is specified and the file should exist but doesn't, or when one of the directories in a pathname does not exist.

3 ESRCH No such process:

No process can be found corresponding to that specified by *pid* in *kill* or *ptrace*.

4 EINTR Interrupted system call:

An asynchronous signal (such as interrupt or quit), which the user has elected to catch, occurred during a system call. If execution is resumed after processing the signal, it will appear as if the interrupted system call returned this error condition.

5 EIO I/O error:

Some physical I/O error. This error may in some cases occur on a call following the one to which it actually applies.

6 ENXIO No such device or address:

I/O on a special file refers to a subdevice which does not exist, or beyond the limits of the device. It may also occur when, for example, a tape drive is not on-line or no disk pack is loaded on a drive.

7 E2BIG Arg list too long:

An argument list longer than 5,120 bytes is presented to a member of the *exec* family.

8 ENOEXEC Exec format error:

A request is made to execute a file which, although it has the appropriate permissions, does not start with a valid magic number (see *a.out(F)*).

9 EBADF Bad file number:

Either a file descriptor refers to no open file, or a read (respectively write) request is made to a file which is open only for writing (respectively reading).

10 ECHILD No child processes:

A *wait* was executed by a process that had no existing or unwaited-for child processes.

- 11 EAGAIN No more processes:  
A *fork* failed because the system's process table is full or the user is not allowed to create any more processes.
- 12 ENOMEM Not enough space:  
During an *exec*, or *sbrk*, a program asks for more space than the system is able to supply. This is not a temporary condition; the maximum space size is a system parameter. The error may also occur if the arrangement of text, data, and stack segments requires too many segmentation registers, or if there is not enough swap space during a *fork*.
- 13 EACCES Permission denied:  
An attempt was made to access a file in a way forbidden by the protection system.
- 14 EFAULT Bad address:  
The system encountered a hardware fault in attempting to use an argument of a system call.
- 15 ENOTBLK Block device required:  
A nonblock file was mentioned where a block device was required, e.g., in *mount*.
- 16 EBUSY Device busy:  
An attempt to mount a device that was already mounted or an attempt was made to dismount a device on which there is an active file (open file, current directory, mounted-on file, active text segment). It will also occur if an attempt is made to enable accounting when it is already enabled.
- 17 EEXIST File exists:  
An existing file was mentioned in an inappropriate context, e.g., *link*.
- 18 EXDEV Cross-device link:  
A link to a file on another device was attempted.
- 19 ENODEV No such device:  
An attempt was made to apply an inappropriate system call to a device; e.g., read a write-only device.
- 20 ENOTDIR Not a directory:  
A nondirectory was specified where a directory is required, for example, in a path prefix or as an argument to *chdir*(S).
- 21 EISDIR Is a directory:  
An attempt to write on a directory.
- 22 EINVAL Invalid argument:  
An invalid argument (e.g., dismounting a nonmounted device; mentioning an undefined signal in *signal* or *kill*; reading or writing



a file for which *lseek* has generated a negative pointer). Also set by the math functions described in the (S) entries of this manual.

- 23 ENFILE File table overflow:  
The system's table of open files is full and temporarily no more *opens* can be accepted.
- 24 EMFILE Too many open files:  
No process may have more file descriptors open at a time than the number set in the NOFILES tunable kernel parameter.
- 25 ENOTTY Not a character device  
The device requested could not be opened for character I/O.
- 26 ETXTBSY Text file busy:  
An attempt to execute a pure-procedure program which is currently open for writing (or reading). Also an attempt to open for writing a pure-procedure program that is being executed.
- 27 EFBIG File too large:  
The size of a file exceeded the maximum file size (1,082,201,088 bytes) or ULIMIT.
- 28 ENOSPC No space left on device:  
During a *write* to an ordinary file, there is no free space left on the device.
- 29 EPIPE Illegal seek:  
An *lseek* was issued to a pipe.
- 30 EROFS Read-only file system:  
An attempt to modify a file or directory was made on a device mounted read-only.
- 31 EMLINK Too many links:  
An attempt to make more than the maximum number of links (1000) to a file.
- 32 EPIPE Broken pipe:  
A write on a pipe for which there is no process to read the data. This condition normally generates a signal; the error is returned if the signal is ignored.
- 33 EDOM Math arg out of domain of func:  
The argument of a function in the math package is out of the domain of the function.
- 34 ERANGE Math result not representable:  
The value of a function in the math package is not representable within machine precision.



- 35 ENOMSG No message of desired type:  
An attempt was made to receive a message of a type that does not exist on the specified message queue; see *msgop*(S).
- 36 EIDRM Identifier removed:  
This error is returned to a process that resumes execution due to the removal of an identifier from the file system's name space; see *msgctl*(S), *semctl*(S), and *shmctl*(S).
- 37 ECHRNG Channel number out of range:
- 38 E2NSYNC Level 2 not synchronized:
- 39 EL3HLT Level 3 halted:
- 40 EL3RST Level 3 reset:
- 41 ELNRNG Link number out of range:
- 42 EUNATCH Protocol driver not attached:
- 43 ENOCSI No CSI structure available:
- 44 EL2HLT Level 2 halted:
- 45 EDEADLK Deadlock situation detected & avoided:  
A deadlock situation was detected and avoided. This error pertains to file and record locking.
- 46 ENOLCK No record locks available:  
The system's lock table was full, and a file locking or unlocking operation was attempted which would have created an additional lock table entry.
- 47 ERROR 47
- 48 ERROR 48
- 49 ERROR 49
- 50 EBADE Bad exchange descriptor:
- 51 EBADR Bad request descriptor:
- 52 EXFULL Exchange table full:
- 53 ENANO Anode table overflow:
- 54 EBADRQC Bad request code:

- 55 EBADSLT Invalid slot:
- 56 EDEADLOCK File locking deadlock:
- 57 EBFONT Bad font file fmt:
- 58 ERROR 58
- 59 ERROR 59
- 60 ENOSTR Device not a stream:  
A *putmsg(S)* or *getmsg(S)* system call was attempted on a file descriptor that is not a STREAMS device.
- 61 ENODATA No data:
- 62 ETIME Timer expired:  
The timer set for a STREAMS *ioctl(S)* call has expired. The cause of this error is device specific and could indicate either a hardware or software failure, or perhaps a timeout value that is too short for the specific operation. The status of the *ioctl(S)* operation is indeterminate.
- 63 ENOSR Out of streams resources:  
During a STREAMS *open(S)*, either no STREAMS queues or no STREAMS head data structures were available.
- 64 ENONET Machine is not on the network:  
This error is Remote File Sharing (RFS)-specific. It occurs when users try to advertise, unadvertise, mount, or unmount remote resources while the machine has not done the proper start-up to connect to the network.
- 65 ENOPKG Package not installed:  
This error occurs when users attempt to use a system call from a package which has not been installed.
- 66 EREMOTE The object is remote:  
This error is RFS-specific. It occurs when users try to advertise a resource which is not on the local machine, or try to mount/unmount a device (or path name) that is on a remote machine.
- 67 ENOLINK The link has been severed:  
This error is RFS-specific. It occurs when the link (virtual circuit) connecting to a remote machine is gone.)
- 68 EADV Advertise error:  
This error is RFS-specific. It occurs when users try to advertise a resource which has been advertised already, or try to stop the RFS while there are resources still advertised, or try to force unmount a resource when it is still advertised.

- 69 ESRMNT *srmount* error:  
This error is RFS-specific. It occurs when users try to stop RFS while there are resources still mounted by remote machines.
- 70 ECOMM Communication error on send:  
This error is RFS-specific. It occurs when trying to send messages to remote machines but no virtual circuit can be found.
- 71 EPROTO Protocol error:  
Some protocol error occurred. This error is device-specific, but is generally not related to a hardware failure.
- 72 ERROR 72
- 73 ERROR 73
- 74 EMULTIHOP Multihop attempted:  
This error is RFS-specific. It occurs when users try to access remote resources which are not directly accessible.
- 75 ELBIN Undefined:
- 76 EDOTDOT Undefined:
- 77 EBADMSG Not a data message:  
During a *read(S)*, *getmsg(S)*, or *ioctl(S)* *I\_RECVFD* system call to a STREAMS device, something has come to the head of the queue that can't be processed. That something depends on the system call:  
*read(S)* – control information or a passed file descriptor.  
*getmsg(S)* – passed file descriptor.  
*ioctl(S)* – control or data information.
- 78 ENAMETOOLONG Filename too long:  
The filename given was longer than is allowed.
- 79 ERROR 79
- 80 ENOTUNIQ Name not unique on network:  
Each system on the network must have a unique name.
- 81 EBADFD File descriptor in bad state:
- 82 EREMCHG Remote address changed:
- 83 ELIBACC Cannot access a needed shared lib:  
Trying to *exec(S)* an *a.out* that requires a shared library (to be linked in) and the shared library doesn't exist or the user doesn't have permission to use it.



- 84 ELIBBAD Accessing a corrupted shared lib:  
Trying to *exec(S)* an *a.out* that requires a shared library (to be linked in) and *exec(S)* could not load the shared library. The shared library is probably corrupted.
- 85 ELIBSCN .lib section in *a.out* corrupted:  
Trying to *exec(S)* an *a.out* that requires a shared library (to be linked in) and there was erroneous data in the .lib section of the *a.out*. The .lib section tells *exec(S)* what shared libraries are needed. The *a.out* is probably corrupted.
- 86 ELIBMAX Attempting to link in more shared libraries than system limit:  
Trying to *exec(S)* an *a.out* that requires more shared libraries (to be linked in) than is allowed on the current configuration of the system. See the *System Administrator's Guide*.
- 87 ELIBEXEC Cannot exec a shared library directly:  
Trying to exec a shared library directly. This is not allowed.
- 88 ERROR 88
- 89 ENOSYS Function not implemented:
- Error messages 90 through 134 are not defined in the standard kernel but are sometimes used by additional software.
- 135 EUCLEAN File system needs cleaning:  
An attempt was made to *mount(S)* a file system whose super-block is not flagged clean.
- 136 ERROR 136
- 137 ENOTNAM Not a name file:  
A *creatsem(S)*, *opensem(S)*, *waitsem(S)*, or *sigsem(S)* was issued using an invalid semaphore identifier.
- 138 ENAVAIL Not available:  
An *opensem(S)*, *waitsem(S)* or *sigsem(S)* was issued to a semaphore that has not been initialized by a call to *creatsem(S)*. A *sigsem* was issued to a semaphore out of sequence; i.e., before the process has issued the corresponding *waitsem* to the semaphore. An *nwaitsem* was issued to a semaphore guarding a resource that is currently in use by another process. The semaphore on which a process was waiting has been left in an inconsistent state when the process controlling the semaphore exits without relinquishing control properly; i.e., without issuing a *waitsem* on the semaphore.
- 139 EISNAM Is a name file:  
A name file (semaphore, shared data, etc.) was specified when not expected.

- 140 EREMOTEIO Remote i/o error:  
There was an i/o error on a remote device.
- 141 EINIT reserved:  
This error number is reserved for future use.
- 142 EREMDEV reserved  
This error number is reserved for future use.
- 143 ERROR 143
- 144 ERROR 144
- 145 ENOTEMPTY Directory not empty

## Definitions

---

### *Process ID*

Each active process in the system is uniquely identified by a positive integer called a process ID. The range of this ID is from 0 to 30,000.

### *Parent Process ID*

A new process is created by a currently active process; see *fork(S)*. The parent process ID of a process is the process ID of its creator.

### *Process Group ID*

Each active process is a member of a process group that is identified by a positive integer called the process group ID. This ID is the process ID of the group leader. This grouping permits the signaling of related processes; see *kill(S)*.

### *Process Group Leader*

A process group leader is any process whose process group ID is the same as its process ID. Any process may become a group leader by calling *setgrp(S)*. A process inherits the process group ID of the process that created it, see *fork(S)* and *exec(S)*.

### *TTY Group ID*

Each active process can be a member of a terminal group that is identified by a positive integer called the TTY group ID. This

grouping is used to terminate a group of related process upon termination of one of the processes in the group; see *exit(S)* and *signal(S)*.

### *Real User ID and Real Group ID*

Each user allowed on the system is identified by a positive integer called a real user ID.

Each user is also a member of a group. The group is identified by a positive integer called the real group ID.

An active process has a real user ID and a real group ID that are set to the real user ID and real group ID, respectively, of the user responsible for the creation of the process.

### *Effective User ID and Effective Group ID*

An active process has an effective user ID and an effective group ID that are used to determine file access permissions (see below). The effective user ID and effective group ID are equal to the process' real user ID and real group ID respectively, unless the process or one of its ancestors evolved from a file that had the set-user-ID bit or set-group ID bit set; see *exec(S)*.

### *Super-User*

A process is recognized as a *super-user* process and is granted special privileges if its effective user ID is 0.

### *Special Processes*

The processes with a process ID of 0 and a process ID of 1 are special processes and are referred to as *proc0* and *proc1*.

*proc0* is the scheduler. *proc1* is the initialization process (*init*). *Proc1* is the ancestor of every other process in the system and is used to control the process structure.

### *Filename*

Names consisting of up to 14 characters may be used to name an ordinary file, special file or directory.

These characters may be selected from the set of all character values excluding 0 (null) and the ASCII code for a slash (/).



Note that it is generally unwise to use \*, ?, [, or ] as part of filenames because of the special meaning attached to these characters by the shell. Likewise, the high order bit of the character should not be set.

### *Pathname and Path Prefix*

A pathname is a null-terminated character string starting with an optional slash (/), followed by zero or more directory names separated by slashes, optionally followed by a filename. A filename is a string of 1 to 14 characters other than the ASCII slash and null, and a directory name is a string of 1 to 14 characters (other than the ASCII slash and null) naming a directory.

If a pathname begins with a slash, the path search begins at the *root* directory. Otherwise, the search begins from the current working directory.

A slash by itself names the root directory.

Unless specifically stated otherwise, the null pathname is treated as if it named a nonexistent file.

### *Directory*

Directory entries are called links. By convention, a directory contains at least two links, . and .., referred to as “dot” and “dot-dot” respectively. Dot refers to the directory itself and dot-dot refers to its parent directory.

### *Root Directory and Current Working Directory*

Each process has a concept of a root directory and a current working directory for the purpose of resolving pathname searches associated with it. A process' root directory need not be the root directory of the root file system. See *chroot* (ADM) and *chroot* (S).

### *File Access Permissions*

Read, write, and execute/search permissions on a file are granted to a process if one or more of the following are true:

- The process' effective user ID is super-user.

- The process' effective user ID matches the user ID of the owner of the file and the appropriate access bit of the “owner” portion (0700) of the file mode is set.

The process' effective user ID does not match the user ID of the owner of the file, and the process' group ID matches the group of the file, and the appropriate access bit of the "group" portion (070) of the file mode is set.

The process' effective user ID does not match the user ID of the owner of the file, and the process' effective group ID does not match the group ID of the file, and the appropriate access bit of the "other" portion (07) of the file mode is set.

Otherwise, the corresponding permissions are denied. See *chmod*(C) and *chmod*(S).

### Message Queue Identifier

A message queue identifier (*msqid*) is a unique positive integer created by a *msgget*(S) system call. Each *msqid* has a message queue and a data structure associated with it. The data structure is referred to as *msqid\_ds* and contains the following members:

```

struct ipc_perm msg_perm; /* operation permission struct */
struct msg *msg_first; /* ptr to first message on q */
struct msg *msg_last; /* ptr to last message on q */
ushort msg_cbytes; /* current number of bytes on q */
ushort msg_qnum; /* number of msgs on q */
ushort msg_qbytes; /* max number of bytes on q */
ushort msg_lspid; /* pid of last msgsnd operation */
ushort msg_lrpid; /* pid of last msgrcv operation */
time_t msg_stime; /* last msgsnd time */
time_t msg_rtime; /* last msgrcv time */
time_t msg_ctime; /* last change time */
/* Times measured in secs since */
/* 00:00:00 GMT, Jan. 1, 1970 */

```

**msg\_perm** is an *ipc\_perm* structure that specifies the message operation permission (see below). The structure includes the following members:

```

ushort uid; /* owner's user id */
ushort gid; /* owner's group id */
ushort cuid; /* creator's user id */
ushort cgid; /* creator's group id */
ushort mode; /* r/w permission */
ushort seq; /* slot usage sequence number */
key_t key; /* key */

```

**msg\_qnum** is the number of messages currently on the queue. **msg\_qbytes** is the maximum number of bytes allowed on the queue. **msg\_lspid** is the process ID of the last process that performed a *msgsnd* operation. **msg\_lrpid** is the process ID of the last process that performed a *msgrcv* operation. **msg\_stime** is the time of the last



*msgsnd* operation, **msg\_rtime** is the time of the last *msgrcv* operation, and **msg\_ctime** is the time of the last *msgctl*(S) operation that changed a member in the above structure.

### Message Operation Permissions

In the *msgop*(S) and *msgctl*(S) system call descriptions, the permission required for an operation is given as “{token}”, where “token” is the type of permission needed. It is interpreted as follows:

|       |                       |
|-------|-----------------------|
| 00400 | Read by user          |
| 00200 | Write by user         |
| 00060 | Read, write by group  |
| 00006 | Read, write by others |

Read and write permissions on a *msqid* are granted to a process if one or more of the following are true:

The effective user ID of the process is super-user.

The effective user ID of the process matches **msg\_perm.uid** or **msg\_perm.cuid** in the data structure associated with *msqid*, and the appropriate bit of the “user” portion (0600) of **msg\_perm.mode** is set.

The effective user ID of the process does not match **msg\_perm.uid** or **msg\_perm.cuid** and the effective group ID of the process matches **msg\_perm.gid** or **msg\_perm.cgid** and the appropriate bit of the “group” portion (060) of **msg\_perm.mode** is set.

The effective user ID of the process does not match **msg\_perm.uid** or **msg\_perm.cuid** and the effective group ID of the process does not match **msg\_perm.gid** or **msg\_perm.cgid** and the appropriate bit of the “other” portion (06) of **msg\_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

### Semaphore Identifier

A semaphore identifier (*semid*) is a unique positive integer created by a *semget*(S) system call. Each *semid* has a set of semaphores and a data structure associated with it. The data structure is referred to as *semid\_ds* and contains the following members:



```

struct ipc_perm sem_perm; /* operation permission struct */
struct sem *sem_base; /* ptr to first semaphore in set */
ushort sem_nsems; /* number of sems in set */
time_t sem_otime; /* last operation time */
time_t sem_ctime; /* last change time */
/* Times measured in secs since */
/* 00:00:00 GMT, Jan. 1, 1970 */

```

**sem\_perm** is an **ipc\_perm** structure that specifies the semaphore operation permission (see below). This structure includes the following members:

```

ushort cuid; /* creator user id */
ushort cgid; /* creator group id */
ushort uid; /* user id */
ushort gid; /* group id */
ushort mode; /* r/a permission */

```

The value of **sem\_nsems** is equal to the number of semaphores in the set. Each semaphore in the set is referenced by a positive integer referred to as a "sem\_num". Sem\_num values run sequentially from 0 to the value of **sem\_nsems** minus 1. **sem\_otime** is the time of the last **semop(S)** operation, and **sem\_ctime** is the time of the last **semctl(S)** operation that changed a member of the above structure.

A semaphore is a data structure that contains the following members:

```

ushort semval; /* semaphore value */
short sempid; /* pid of last operation */
ushort semncnt; /* # awaiting semval > cval */
ushort semzcnt; /* # awaiting semval = 0 */

```

**semval** is a non-negative unsigned short. **sempid** is equal to the process ID of the last process that performed a semaphore operation on this semaphore. **semncnt** is a count of the number of processes that are currently suspended awaiting this semaphore's **semval** to become greater than its current value. **semzcnt** is a count of the number of processes that are currently suspended awaiting this semaphore's **semval** to become zero.

### Semaphore Operation Permissions

In the **semop(S)** and **semctl(S)** system call descriptions, the permission required for an operation is given as "{token}", where "token" is the type of permission needed and is interpreted as follows:

|       |                       |
|-------|-----------------------|
| 00400 | Read by user          |
| 00200 | Alter by user         |
| 00060 | Read, alter by group  |
| 00006 | Read, alter by others |

Read and alter permissions for a *semid* are granted to a process if one or more of the following are true:

The effective user ID of the process is super-user.

The effective user ID of the process matches **sem\_perm.uid** or **sem\_perm.cuid** in the data structure associated with *semid*, and the appropriate "user" portion (0600) bit of **sem\_perm.mode** is set.

The effective user ID of the process does not match **sem\_perm.uid**, or **sem\_perm.cuid** and the effective group ID of the process matches **sem\_perm.gid** or **sem\_perm.cgid** and the appropriate bit of the "group" portion (060) of **sem\_perm.mode** is set.

The effective user ID of the process does not match **sem\_perm.uid** or **sem\_perm.cuid** and the effective group ID of the process does not match **sem\_perm.gid** or **sem\_perm.cgid** and the appropriate bit of the "other" portion (06) of **sem\_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

### Shared Memory Identifier

A shared memory identifier (*shmid*) is a unique positive integer created by a *shmget*(S) system call. Each *shmid* has a segment of memory (referred to as a shared memory segment) and a data structure associated with it. The data structure is referred to as *shmid\_ds* and contains the following members:

```

struct ipc_perm shm_perm; /* operation permission struct */
int shm_segsz; /* size of segment */
ushort shm_cpid; /* creator pid */
ushort shm_lpid; /* pid of last operation */
short shm_nattch; /* number of current attaches */
time_t shm_atime; /* last attach time */
time_t shm_dtime; /* last detach time */
time_t shm_ctime; /* last change time */
/* Times measured in secs since */
/* 00:00:00 GMT, Jan. 1, 1970 */

```

**shm\_perm** is an *ipc\_perm* structure that specifies the shared memory operation permission (see below). The structure includes the following members:



```

ushort cuid; /* creator user id */
ushort cgid; /* creator group id */
ushort uid; /* user id */
ushort gid; /* group id */
ushort mode; /* r/w permission */

```

**shm\_segsz** specifies the size of the shared memory segment. **shm\_cpid** is the process ID of the process that created the shared memory identifier. **shm\_lpid** is the process ID of the last process that performed a *shmop*(S) operation: **shm\_nattch** is the number of processes that currently have this segment attached. **shm\_atime** is the time of the last *shmat* operation. **shm\_dtime** is the time of the last *shmdt* operation, and **shm\_ctime** is the time of the last *shmctl*(S) operation that changed one of the above structure members.

### Shared Memory Operation Permissions

In the *shmop*(S) and *shmctl*(S) system call descriptions, the permission required for an operation is given as "{token}", where "token" is the type of permission needed. It is interpreted as follows:

|       |                       |
|-------|-----------------------|
| 00400 | Read by user          |
| 00200 | Write by user         |
| 00060 | Read, write by group  |
| 00006 | Read, write by others |

Read and write permissions on a shmid are granted to a process if one or more of the following are true:

The effective user ID of the process is super-user.

The effective user ID of the process matches **shm\_perm.uid** or **shm\_perm.cuid** in the data structure associated with *shmid* and the appropriate bit of the "user" portion (0600) of **shm\_perm.mode** is set.

The effective user ID of the process does not match **shm\_perm.uid** or **shm\_perm.cuid** and the effective group ID of the process matches **shm\_perm.gid** or **shm\_perm.cgid** and the appropriate bit of the "group" portion (060) of **shm\_perm.mode** is set.

The effective user ID of the process does not match **shm\_perm.uid** or **shm\_perm.cuid** and the effective group ID of the process does not match **shm\_perm.gid** or **shm\_perm.cgid** and the appropriate bit of the "other" portion (06) of **shm\_perm.mode** is set.

Otherwise, the corresponding permissions are denied.



## **See Also**

---

`close(S)`, `ioctl(S)`, `open(S)`, `pipe(S)`, `read(S)`, `write(S)`

## **a64l, l64a**

convert between long integer and base-64 ASCII string

### **Syntax**

---

```
long a64l (s)
char *s;
```

```
char *l64a (l)
long l;
```

### **Description**

---

These functions are used to maintain numbers stored in *base-64* ASCII characters. This is a notation by which long integers can be represented by up to six characters; each character represents a "digit" in a radix-64 notation.

The characters used to represent "digits" are . for 0, / for 1, 0 through 9 for 2-11, A through Z for 12-37, and a through z for 38-63.

The *a64l* function takes a pointer to a null-terminated base-64 representation and returns a corresponding **long** value. If the string pointed to by *s* contains more than six characters, *a64l* will use the first six.

The *a64l* function scans the character string from left to right, decoding each character as a 6-bit Radix 64 number.

The *l64a* function takes a **long** argument and returns a pointer to the corresponding base-64 representation. If the argument is 0, *l64a* returns a pointer to a null string.

### **Note**

---

The value returned by *l64a* is a pointer into a static buffer, the contents of which are overwritten by each call.

### **Standards Conformance**

---

*a64l* and *l64a* are conformant with:

AT&T SVID Issue 2, Select Code 307-127.

## abort

---

generate an abort fault

### Syntax

---

`int abort ( )`

### Description

---

The *abort* function does the work of *exit*(S), but instead of just exiting, *abort* causes **SIGABRT** to be sent to the calling process. If **SIGABRT** is neither caught nor ignored, all *stdio*(S) streams are flushed prior to the signal being sent, and a core dump results.

The *abort* function returns the value of the *kill*(S) system call.

### See Also

---

*sdb*(CP), *exit*(S), *kill*(S), *signal*(S).

### Diagnostics

---

If **SIGABRT** is neither caught nor ignored, and the current directory is writable, a core dump is produced and the message "abort - core dumped" is written by the shell.

### Standards Conformance

---

*abort* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.



## **abs**

---

return integer absolute value

### **Syntax**

---

```
int abs (i)
int i;
```

### **Description**

---

The *abs* function returns the absolute value of its integer operand.

### **See Also**

---

*floor*(S).

### **Note**

---

In two's-complement representation, the absolute value of the negative integer with largest magnitude is undefined. Some implementations trap this error, but others simply ignore it.

### **Standards Conformance**

---

*abs* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

## acceptable\_password

---

determine if password is cryptic

### Syntax

---

```
#include <sys/types.h>
#include <sys/security.h>
#include <sys/audit.h>
#include <prot.h>
```

```
int acceptable_password (word, stream)
char *word;
FILE *stream;
```

### Description

---

This routine determines if the given password is reasonable, i.e., that it is very hard to deduce from a number of heuristic means. The clear-text (plaintext) password is passed as the first argument and the file pointer of the stream to report failure reasons is the second argument. If this checking is to be silent, the second argument should be the NULL file pointer.

When *acceptable\_password* returns a 1, the password provided meets all the tests below. When it returns a 0, the password failed to meet at least one of the tests.

The selectivity criteria for the password include but may not be limited to the following four tests:

#### Palindrome

This test passes if the word is not a palindrome. (A palindrome is spelled the same backwards as forwards.) Examples of palindromes that fail on this test are: mom, dad, noon, redivider, radar. Palindromes do not make good passwords because they reduce an  $n$  character password to  $n/2 + 1$  characters. A penetrator knowing that palindromes were legal could use heuristics that could deduce the password much more quickly than if they were excluded.

#### Login Name

This test passes if the password is not a login name for the system. The many insecure systems allow passwords to be the login name itself. This is a fact known by many penetrators. All login names are

excluded because a user that is the owner of several pseudo-user accounts may elect to use the login name of one account as the password for all his accounts.

**Group Name** Similar to the login name issue, this test passes if the password is not a group name.

**English word** This test passes if the *spell*(C) determines that this is not an English word. A penetrator then could not search the on-line dictionary to find the password. The *spell* program also has some built-in rules that go beyond the actual on-line dictionary in determining what is a proper word, and this routine takes advantage of that.

## Notes

---

This routine only works as advertised when *set\_auth\_parameters* is called as the first item in *main*().

## Files

---

/etc/passwd  
/etc/group  
/usr/lib/spell/\*

## See Also

---

*spell*(C), *getpwent*(S), *getgrent*(S)

## Value Added

---

*acceptable\_password* is an extension of AT&T System V provided by the Santa Cruz Operation.



## access

---

determine accessibility of a file

### Syntax

---

```
#include <unistd.h>
```

```
int access (path, amode)
char *path;
int amode;
```

```
int eaccess(path, amode)
char *path;
int amode;
```

### Description

---

The *path* argument points to a path name naming a file. The *access* function checks the named file for accessibility according to the bit pattern contained in *amode*, using the real user ID in place of the effective user ID and the real group ID in place of the effective group ID. *eaccess* is the same as *access* except that the effective user ID and group ID are used. The bit pattern contained in *amode* is constructed as follows:

|    |                         |
|----|-------------------------|
| 04 | read                    |
| 02 | write                   |
| 01 | execute (search)        |
| 00 | check existence of file |

The symbolic constants for the argument **amode** are defined by the `<unistd.h>` header file and are as follows:

| Name        | Description                                  |
|-------------|----------------------------------------------|
| <b>R_OK</b> | test for <i>read</i> permission.             |
| <b>W_OK</b> | test for <i>write</i> permission.            |
| <b>X_OK</b> | test for <i>execute (search)</i> permission. |
| <b>F_OK</b> | test for existence of file.                  |

The argument **amode** is either the logical OR of one or more of the values of the symbolic constants for **R\_OK**, **W\_OK**, and **X\_OK** or is the value of the symbolic constant **F\_OK**.

Access to the file is denied if one or more of the following is true:

|             |                                                                                           |
|-------------|-------------------------------------------------------------------------------------------|
| [ENOTDIR]   | A component of the path prefix is not a directory.                                        |
| [ENOENT]    | Read, write, or execute (search) permission is requested for a null path name.            |
| [ENOENT]    | The named file does not exist.                                                            |
| [EACCES]    | Search permission is denied on a component of the path prefix.                            |
| [EROFS]     | Write access is requested for a file on a read-only file system.                          |
| [ETXTBSY]   | Write access is requested for a pure procedure (shared text) file that is being executed. |
| [EACCES]    | Permission bits of the file mode do not permit the requested access.                      |
| [EFAULT]    | <i>path</i> points outside the allocated address space for the process.                   |
| [EINTR]     | A signal was caught during the <i>access</i> system call.                                 |
| [ENOLINK]   | <i>path</i> points to a remote machine and the link to that machine is no longer active.  |
| [EMULTIHOP] | Components of <i>path</i> require hopping to multiple remote machines.                    |

The owner of a file has permission checked with respect to the "owner" read, write, and execute mode bits. Members of the file's group other than the owner have permissions checked with respect to the "group" mode bits, and all others have permissions checked with respect to the "other" mode bits.

## See Also

---

chmod(S), stat(S).

## Diagnostics

---

If the requested access is permitted, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## **Standards Conformance**

---

*access* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;

and NIST FIPS 151-1.



## acct

---

enable or disable process accounting

### Syntax

---

```
int acct (path)
char *path;
```

### Description

---

*acct* is used to enable or disable the system process accounting routine. If the routine is enabled, an accounting record will be written on an accounting file for each process that terminates. Termination can be caused by one of two things: an *exit* call or a signal (see *exit*(S) and *signal*(S)). The effective user ID of the calling process must be super-user to use this call.

*path* points to a pathname naming the accounting file. The accounting file format is given in *acct*(F).

The accounting routine is enabled if *path* is non-zero and no errors occur during the system call. It is disabled if *path* is zero and no errors occur during the system call.

*acct* will fail if one or more of the following is true:

|           |                                                                           |
|-----------|---------------------------------------------------------------------------|
| [EPERM]   | The effective user of the calling process is not super-user.              |
| [EBUSY]   | An attempt is being made to enable accounting when it is already enabled. |
| [ENOTDIR] | A component of the path prefix is not a directory.                        |
| [ENOENT]  | One or more components of the accounting file path name do not exist.     |
| [EACCES]  | The file named by <i>path</i> is not an ordinary file.                    |
| [EROFS]   | The named file resides on a read-only file system.                        |
| [EFAULT]  | <i>Path</i> points to an illegal address.                                 |

### See Also

---

*exit*(S), *signal*(S), *acct*(F).

## Diagnostics

---

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## Standards Conformance

---

*acct* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## alarm

---

set a process alarm clock

### Syntax

---

**unsigned alarm (sec)**  
**unsigned sec;**

### Description

---

The *alarm* system call instructs the alarm clock of the calling process to send the signal **SIGALRM** to the calling process after the number of real time seconds specified by *sec* have elapsed [see *signal(S)*].

Alarm requests are not stacked; successive calls reset the alarm clock of the calling process.

If *sec* is 0, any previously made alarm request is canceled. The *fork(S)* system call sets the alarm clock of a new process to 0. A process created by the *exec(S)* family of calls inherits the time left on the old process's alarm clock.

### See Also

---

*exec(S)*, *fork(S)*, *pause(S)*, *signal(S)*, *sigset(S)*.

### Diagnostics

---

The *alarm* system call returns the amount of time previously remaining in the alarm clock of the calling process.

### Standards Conformance

---

*alarm* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.



## assert

---

verify program assertion

### Syntax

---

```
#include <assert.h>
```

```
assert (expression)
int expression;
```

### Description

---

This macro is useful for putting diagnostics into programs. When it is executed, if *expression* is false (zero), *assert* prints

“Assertion failed: *expression*, file *xyz*, line *nnn*”

on the standard error output and aborts. In the error message, *xyz* is the name of the source file and *nnn* the source line number of the *assert* statement.

Compiling with the preprocessor option **-DNDEBUG** (see *cpp* (CP)), or with the preprocessor control statement “**#define NDEBUG**” ahead of the “**#include <assert.h>**” statement, will stop assertions from being compiled into the program.

### See Also

---

*cpp*(CP), *abort*(S).

### Notes

---

Since *assert* is implemented as a macro, the *expression* may not contain any string literals.

### Standards Conformance

---

*assert* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
and ANSI X3.159-198X C Language Draft Standard, May 13, 1988.

## atexit

---

calls a process at termination

### Syntax

---

```
#include <stdlib.h>
```

```
int atexit(func)
void (*func)(void);
```

### Description

---

The *atexit* function is passed the address of a function (*func*) to be called when the program terminates normally. Successive calls to *atexit* create a register of functions that are executed “last in, first out.” No more than 32 functions can be registered with *atexit*. The functions passed to *atexit* cannot take parameters.

### Return Value

---

*atexit* returns a pointer to the function if successful, or NULL if there is no space left to store the function pointer.

### See Also

---

abort(S), exit(S)

### Example

---

```
#include <stdlib.h>
#include <stdio.h>

main()
{
 int fn1(), fn2(), fn3(), fn4();
 atexit(fn1);
 atexit(fn2);
 atexit(fn3);
 atexit(fn4);
 printf("This is executed first.\n");
}

int fn1()
{
 printf("next.\n");
}
```

```
int fn2()
{
 printf("executed ");
}

int fn3()
{
 printf("is ");
}

int fn4()
{
 printf("This ");
}
```

Output:

This is executed first.  
This is executed next.

This program pushes four functions onto the stack of functions to be executed when *atexit* is called. When the program exits, these programs are executed on a last-in, first-out basis.

## Standards Conformance

---

*atexit* is conformant with:

ANSI X3.159-198X C Language Draft Standard, May 13, 1988.



## atof, atoi, atol

---

converts ASCII to numbers

### Syntax

---

```
#include <math.h>
double atof (nptr)
char *nptr;
```

```
#include <stdlib.h>
int atoi (nptr)
char *nptr;
```

```
long atol (nptr)
char *nptr;
```

### Description

---

These functions convert a string pointed to by *nptr* to floating, integer, and long integer numbers respectively. The first unrecognized character ends the string.

*atof* recognizes a string of the form:

[ + | - ] digits [ . digits ] [ e | E [ + | - ] digits ]

where the digits are contiguous decimal digits. Any number of tabs and spaces may precede the string. The + and - signs are optional. Either e or E may be used to mark the beginning of the exponent.

*atoi* and *atol* recognize strings of the form:

[ + | - ] digits

where the digits are contiguous decimal digits. Any number of tabs and spaces may precede the string. The + and - signs are optional.

### See Also

---

scanf(S)

### Notes

---

There are no provisions for overflow.

These routines must be linked by using the **-lm** linker option.

## **Standards Conformance**

---

*atof*, *atoi* and *atol* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

and ANSI X3.159-198X C Language Draft Standard, May 13, 1988.

## **audit\_open, audit\_read, audit\_close**

open and access audit session data on a record basis

### **Syntax**

---

```
int audit_open(session)
int session;

struct audit_header *audit_read()

int audit_close()
```

### **Description**

---

*Audit\_open* provides an interface for opening an audit data session which has been previously collected on the system. The routine requires one argument, *session*, which indicates the audit session number. This may be acquired from a session list by using the *auditif(1M)* interface program.

Once a session is open, *audit\_read()* may be used to sequentially retrieve audit records from the audit session data files. Each call returns a pointer to the next audit record header which identifies the record size, the record type, the event type, and other audit related information. The actual record formats are defined in *audit(7)*. The function returns a NULL pointer if an error or end-of-file occurs.

Before another session may be accessed, *audit\_close()* must be used to terminate processing for the current session. Another session may then be opened.

### **See Also**

---

authaudit(S) audit(HW), "Programming in a Secure Environment" in the *Programmer's Guide*.

### **DIAGNOSTICS**

---

Upon successful completion, *audit\_open()* and *audit\_close()* return 0. Otherwise, they return -1 with *errno* set to indicate the error. *Audit\_read()* returns a pointer to the next audit record or NULL if an error occurs or EOF is encountered.



**Value Added**

---

*audit\_close*, *audit\_open* and *audit\_read* are extensions of AT&T System V provided by the Santa Cruz Operation.

## authaudit

---

produce audit records due to authentication events

### Syntax

---

```
#include <sys/types.h>
#include <sys/security.h>
#include <sys/audit.h>
#include <prot.h>
```

```
void audit_security_failure (object, expected, curr, action, result)
int object;
long expected;
long curr;
char *action;
char *result;
```

```
void audit_subsystem (subsystem_type, action, result)
int subsystem_type;
char *action;
char *result;
```

```
void audit_auth_entry (desired_entry, type, problem)
char *desired_entry;
int type;
char *problem;
```

```
void audit_no_resource (resource, object, problem)
char *resource;
int object;
char *problem;
```

```
void audit_lax_file (path, problem)
char *path;
char *problem;
```

```
audit_login (pr, pwd, terminal_name, code)
register struct pr_passwd *pr;
register struct passwd *pwd;
register char *terminal_name;
int code;
```

```
audit_passwd (name, code)
char *name;
int code;
```

```
audit_lock (name, code, tries)
char *name;
int code;
int tries;

audit_adjust_mask (pr)
register struct pr_passwd *pr;
```

## Description

---

These routines provide standard interfaces to the secure audit facility from routines and programs that manipulate the Authentication database. Depending on the circumstances (UIDs, privileges), they either write directly to the audit special device `/dev/auditw`, or they pass the information onto the `dlvr_audit` program.

*Audit\_security\_failure* records a problem involving a system *object*, as defined in `<audit.h>`. Even though kernel auditing may have recorded the same security problem as one or a series of failed system calls, this routine will produce an audit record specifically noting the high-level security problem in terms of trusted entity failure. If appropriate, the *expected* and current *curr* values are recorded to further help in diagnosing the problem. The high level *action* attempted and the *result* of the failure are required. This is the means to report a high-level security problem that prevents or impedes the correct operation of a trusted process or subsystem. If the trusted process detects and corrects security problems, the invocation of this routine is the detection component of that mechanism.

*Audit\_subsystem* records an audit record for high-level security events specific to a subsystem as defined by *subsystem\_type* defined in `<audit.h>`. The high-level *action* and either positive or negative *result* is recorded. This is the means to report a problem or significant event in a specific subsystem.

*Audit\_auth\_entry* produces an audit record noting that the name *desired\_entry* has a *problem* in the *type* database, which is a component database (one of: `/etc/passwd`, `/etc/group`, Protected Password database, Terminal Control database, File control database, Command Control database, System Default database, Subsystem database) of the Authentication database. This is the means to report a database inconsistency in an entry of the appropriate database.

*Audit\_no\_resource* prints an audit record that says the *resource* could not be obtained of system type *object*, as defined in `<audit.h>`. The *problem* that results is also recorded. Typically, this is used to denote that a vital resource like memory could not be allocated and a security operation had to be aborted.



*Audit\_lax* file produces an audit record about the file *path* and the exact *problem* that makes the file differ from the File Control database entry describing it. This is the means to report a breakdown of a sanity check on the proper setup of system files.

## Notes

---

These routines only work as advertised when *set\_auth\_parameters* is called as the first item in *main()*.

## Files

---

/dev/auditw

## See Also

---

audit(HW), identity(S).

## Value Added

---

*authaudit* is an extension of AT&T System V provided by the Santa Cruz Operation.

## **authcap**

get information from the authentication database

---

### **Syntax**

---

```
#include <sys/types.h>
#include <sys/security.h>
#include <sys/audit.h>
#include <prot.h>
```

```
int agetuser (user)
char *user;
```

```
int agetfile (filepos, fd)
long *filepos;
int fd;
```

```
int agetcommand (filepos, fd)
long *filepos;
int fd;
```

```
int agetty (filepos, fd)
long *filepos;
int fd;
```

```
int agetdefault (filepos, fd)
long *filepos;
int fd;
```

```
char *agetstr (id, area)
char *id;
char **area;
```

```
int agetnum (id)
char *id;
```

```
int agetflag (id)
char *id;
```

```
void asetdefaults (new_defaults)
int new_defaults;
```

## Description

---

These routines manage ASCII database files in hierarchies as described in *authcap(F)*. All program references to the database should use these routines. Also, these routines are the ones that understand the *authcap(F)* format and are guaranteed to provide the same interface, even if the database format or name designations change.

*Agetuser* finds the entry associated with the user name argument. *Agetcommand* finds the next command entry in the file described by *fd* starting at the file position *filepos* within the file. *Filepos* should point to the beginning of a valid entry or at the end of the file. *Agetfile* finds the next file entry in the file in the same way as *agetcommand*. Similarly, *agettty* finds the next tty entry in the file in the same way as *agetcommand*. Tty names are the components without the */dev* part. Example tty names in the database are **console**, and **tty3**. *Agetdefault* finds the next default entry in the file in the same way as *agetcommand*.

Each of the above routines returns a status indicator. A return value of 1 means the entry was found. A return value of 0 means the entry was not found in the file. The calls with the *filepos* argument may update the position referenced by *filepos*.

Once one of the above routines obtains an entry, the next three routines obtain capabilities from that most recently chosen entry. *Agetnum* returns the number associated with the id argument. It returns -1 if the capability cannot be found. *Agetstr* returns the string associated with the id argument. The place the string goes is referenced by the area argument. Not the area argument is a *pointer* to an allocated string, not merely a string. The 0 pointer ((char \*) 0) is returned when the capability cannot be found. A null string is returned as an empty string (\*\*area == '\0'). *Agetflag* returns the flag associated with the id argument. If the flag is set, 1 is returned. If the flag is not set (the @ attribute appears with the id in the file), 0 is returned. If the capability flag cannot be found at all (different from not set), -1 is returned.

*Asetdefaults* sets the place to find system values. It is set to one of the following values for all system references of the database until it is reset again by *asetdefaults*. Initially it is **NORMAL**, where system values are found in the site-selectable part of the database. The value **WEAK** uses an insecure version, while the value **STRONG** uses a secure version of the system-wide values. When none of these values is used, the value **STRONG** is used.



## Notes

---

*Agetuser*, *agetcommand*, *agetfile*, *agetty* and *agetdefault* make use of static areas for an entry. Successive calls to any of these routines will overwrite that data used by later calls to *agetstr*, *agetnum* and *agetflag*.

A numeric capability of -1 cannot be easily assimilated because -1 is the *agetnum* error indicator.

## See Also

---

*getprpwent*(S), *getprtcent*(S), *getprfient*(S), *getprcment*(S), *fields*(S), *authcap*(F)

## Value Added

---

*authcap* is an extension of AT&T System V provided by the Santa Cruz Operation.

## **bessel: j0, j1, jn, y0, y1, yn**

---

### bessel functions

### Syntax

---

```
#include <math.h>
```

```
double j0 (x)
double x;
```

```
double j1 (x)
double x;
```

```
double jn (n, x)
int n;
double x;
```

```
double y0 (x)
double x;
```

```
double y1 (x)
double x;
```

```
double yn (n, x)
int n;
double x;
```

### Description

---

*j0* and *j1* return Bessel functions of  $x$  of the first kind of orders 0 and 1 respectively. *jn* returns the Bessel function of  $x$  of the first kind of order  $n$ .

*y0* and *y1* return Bessel functions of  $x$  of the second kind of orders 0 and 1 respectively. *yn* returns the Bessel function of  $x$  of the second kind of order  $n$ . The value of  $x$  must be positive.

### See Also

---

matherr(S).

### Diagnostics

---

Non-positive arguments cause *y0*, *y1*, and *yn* to return the value **-HUGE** and to set *errno* to **EDOM**. In addition, a message indicating DOMAIN error is printed on the standard error output.

Arguments too large in magnitude cause  $j0$ ,  $j1$ ,  $y0$ , and  $y1$  to return zero and to set *errno* to **ERANGE**. In addition, a message indicating TLOSS error is printed on the standard error output.

These error-handling procedures may be changed with the function *matherr*(S).

## Standards Conformance

---

$j0$ ,  $j1$ ,  $jn$ ,  $y0$ ,  $y1$  and  $yn$  are conformant with:

AT&T SVID Issue 2, Select Code 307-127.br and The X/Open Portability Guide II of January 1987.



## brk, sbrk

change data segment space allocation

### Syntax

```
int brk (endds)
char *endds;

char *sbrk (incr)
int incr;
```

### Description

The *brk* and *sbrk* system calls are used to change dynamically the amount of space allocated for the calling process's data segment (see *exec(S)*). The change is made by resetting the process's break value and allocating the appropriate amount of space. The break value is the address of the first location beyond the end of the data segment. The amount of allocated space increases as the break value increases. Newly allocated space is set to zero. If, however, the same memory space is reallocated to the same process, its contents are undefined.

The *brk* system call sets the break value to *endds* and changes the allocated space accordingly.

The *sbrk* system call adds *incr* bytes to the break value and changes the allocated space accordingly. *incr* can be negative, in which case the amount of allocated space is decreased.

The *brk* and *sbrk* system calls will fail without making any change in the allocated space if one or more of the following is true:

- |          |                                                                                                                                                                                                                                                         |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [ENOMEM] | Such a change would result in more space being allocated than is allowed by the system-imposed maximum process size (see <i>ulimit(S)</i> ).                                                                                                            |
| [EAGAIN] | Total amount of system memory available for a read during physical IO is temporarily insufficient (see <i>shmop(S)</i> ). This may occur even though the space requested was less than the system-imposed maximum process size (see <i>ulimit(S)</i> ). |

### Return Value

Upon successful completion *brk* returns a value of 0, and *sbrk* returns the old break value. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## See Also

---

`exec(S)`, `shmop(S)`, `ulimit(S)`, `end(S)`.

## Standards Conformance

---

*brk* and *sbrk* are conformant with:

The X/Open Portability Guide II of January 1987.

## brkctl

allocates data in a 286 far segment

### Syntax

```
#include <sys/brk.h>
```

```
char far *brkctl(command, increment, ptr)
int command;
long increment;
char far *ptr;
```

### Description

The *brkctl* system call allocates and deallocates memory in additional data segments in small and middle model 286 programs. In order for the C compiler to make use of the return values in small and middle model programs, *brkctl* must be declared to return a far pointer. To enable the 'far' keyword for small model C programs, the **-Me** option to the compiler must be used. Middle model C programs require the **-Mme** option.

*command* is either **BR\_ARGSEG**, **BR\_NEWSEG**, or **BR\_IMPSEG**.

*increment* is a signed long increment. If positive, it must be less than 64K; if negative, its absolute value must be less than the sum of the total memory in all far segments plus the amount allocated in the near segment after process creation.

*ptr* is used only when *command* is **BR\_ARGSEG**.

If *increment* is positive, *brkctl* returns a far pointer to the base of at least *increment* number of bytes of memory (see box on next page).

If the *command* is **BR\_IMPSEG**, and a negative *increment* causes one or more segments to be freed, the 'segment in question' (see the *Return Values* section) is the last remaining segment that was not freed. **BR\_IMPSEG** implies the use of the last data segment. Unless the process is small or middle model and currently has only one data segment, a positive *increment* that would overflow the last data segment causes a new segment to be allocated.

If the *command* is **BR\_ARGSEG**, the *increment* may not be more negative than the size of the segment. The third argument (*ptr*), is assumed to be a far pointer in all models; the offset portion is never used.



If the *command* is **BR NEWSEG**, the *increment* may not be negative at all. Any memory allocated is guaranteed to be at the base of a new segment.

## Return Value

*brkctl()* almost always returns a far pointer to the base of the affected region, (char far \*)-1 on error.

When the *increment* is greater than 0, the return value is a pointer to the base of the newly allocated memory.

When the *increment* is less than or equal to 0, the return value is a pointer to the first illegal byte in the segment in question (usually the base of the deallocated memory). If that segment is full (exactly 64K bytes), the return value will be a pointer to the base of the next segment (which may or may not exist).

| Command   | Increment | Ptr             | Action                                                                                                                   |
|-----------|-----------|-----------------|--------------------------------------------------------------------------------------------------------------------------|
| BR_ARGSEG | 0         | <valid far ptr> | report on segment                                                                                                        |
| BR_ARGSEG | other     | <valid far ptr> | increment specified segment                                                                                              |
| BR_NEWSEG | 0         | -               | allocate new segment, size = 0                                                                                           |
| BR_NEWSEG | other     | -               | allocate new segment, size = increment                                                                                   |
| BR_IMPSEG | 0         | -               | report on last segment; may free up empty segment(s).                                                                    |
| BR_IMPSEG | other     | -               | increment last segment; on large model (or small and middle model with multiple data segments) may allocate new segment. |

## See Also

cc(CP), ld(CP), machine(M), malloc(S), sbrk(S)

## Example

---

The example of *brkctl* below uses the *BR\_NEWSEG* parameter to allocate space for 20,000 integers in a far data segment (on a 286 machine) and fills this memory with the integers from 1 to 20,000. Remember to compile this program with the “-Me” option.

```
#include <sys/brk.h>

#define FNULL (int far*)0

#ifdef M_I386
 #define FAILURE (int *)-1
#else
 #define FAILURE (int far*)-1
#endif

main()
{
 int i,j;

 #ifdef M_I386
 int *fp, *brkctl();
 else
 int far *fp, far *brkctl(); /* both fars are necessary */
 #endif

 fp=brkctl(BR_NEWSEG,(long)sizeof(int)*20000,FNULL);
 if (fp==FAILURE){
 perror("brkctl failed");
 exit(1);
 }
 for (i=0;i<20000;++i)
 fp[i]=i+1;
 for (i=0;i<20000;++i)
 printf("%d\n",fp[i]);
}
```

## Notes

---

The *brkctl* system call should be used only for dynamically allocating additional segments in 80286 small and middle model programs. All other uses should be avoided in favor of *sbrk(S)*, *malloc(S)*, and other standard system services. The functionality of *brkctl* may change in future releases.

*brkctl* is currently available only in protected mode.

In all models, the 'near' data segment must be the first data segment.

*brkctl* calls with **BR\_IMPSEG** and a negative *increment* that would affect a shared data segment are refused.

## Value Added

---

*brkctl* is an extension of AT&T System V provided by the Santa Cruz Operation.



## **bsearch**

---

binary search a sorted table

### **Syntax**

---

```
#include <search.h>
```

```
char *bsearch ((char *) key, (char *) base, nel, sizeof (*key), com-
par)
unsigned nel;
int (*compar)();
```

### **Description**

---

The *bsearch* function is a binary search routine generalized from Knuth (6.2.1) Algorithm B. It returns a pointer into a table indicating where a datum may be found. The table must be previously sorted in increasing order according to a provided comparison function. *key* points to a datum instance to be sought in the table. *base* points to the element at the base of the table. *nel* is the number of elements in the table. *compar* is the name of the comparison function, which is called with two arguments that point to the elements being compared. The function must return an integer less than, equal to, or greater than zero if the first argument is to be considered less than, equal to, or greater than the second.

### **Example**

---

The example below searches a table containing pointers to nodes consisting of a string and its length. The table is ordered alphabetically on the string in the node pointed to by each entry.

This code fragment reads in strings and either finds the corresponding node and prints out the string and its length, or prints an error message.

```

#include <stdio.h>
#include <search.h>

#define TABSIZE 1000

struct node {
 char *string;
 int length;
};

struct node table[TABSIZE]; /* table to be searched */
.
.
.
{
 struct node *node_ptr, node;
 int node_compare(); /* routine to compare 2 nodes */
 char str_space[20]; /* space to read string into */
 .
 .
 .
 node.string = str_space;
 while (scanf("%s", node.string) != EOF) {
 node_ptr = (struct node *)bsearch((char *)(&node),
 (char *)table, TABSIZE,
 sizeof(struct node), node_compare);
 if (node_ptr != NULL) {
 (void)printf("string = %20s, length = %d\n",
 node_ptr->string, node_ptr->length);
 } else {
 (void)printf("not found: %s\n", node.string);
 }
 }
}

/*
 This routine compares two nodes based on an
 alphabetical ordering of the string field.
*/
int
node_compare(node1, node2)
char *node1, *node2;
{
 return (strcmp(
 ((struct node *)node1)->string,
 ((struct node *)node2)->string));
}

```

## See Also

---

bsearch(S), lsearch(S), qsort(S), tsearch(S).

## Diagnostics

---

A NULL pointer is returned if the key cannot be found in the table.

## Notes

---

The pointers to the key and the element at the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

Although *bsearch* is declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

## Standards Conformance

---

*bsearch* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.



# cfspeed: cfgetispeed, cfgetospeed, cfsetispeed, cfsetospeed

baud rate functions

## Syntax

```
#include <termios.h>

speed_t cfgetospeed (termios_p)
struct termios *termios_p;

int cfsetospeed (termios_p, speed)
struct termios *termios_p;
speed_t speed;

speed_t cfgetispeed (termios_p)
struct termios *termios_p;

int cfsetispeed (termios_p, speed)
struct termios *termios_p;
speed_t speed;
```

## Description

The following interfaces are provided for getting and setting the values of the input and output baud rates in the *termios* structure. The effects on the terminal device described below do not become effective until the *tcsetattr* () function is called successfully.

The input and output baud rates are stored in the *termios* structure. The values shown in the following table are supported. The name symbols in this table are defined in *<termios.h>*.

| Name | Description | Name   | Description |
|------|-------------|--------|-------------|
| BO   | Hang up     | B600   | 600 baud    |
| B50  | 50 baud     | B1200  | 1200 baud   |
| B75  | 75 baud     | B1800  | 1800 baud   |
| B110 | 110 baud    | B2400  | 2400 baud   |
| B134 | 134.5 baud  | B4800  | 4800 baud   |
| B150 | 150 baud    | B9600  | 9600baud    |
| B200 | 200 baud    | B19200 | 19200 baud  |
| B300 | 300 baud    | B38400 | 38400 baud  |

The type *speed\_t* shall be defined in *<termios.h>* and shall be an unsigned integral type.

The *termios\_p* argument is a pointer to a *termios* structure.

*cfgetospeed()* returns the output baud rate stored in the *termios* structure pointed to by *termios\_p*.

*cfsetospeed()* sets the output baud rate stored in the *termios* structure pointed to by *termios\_p* to *speed*. The zero baud rate, BO, is used to terminate the connection. If BO is specified, the modem control lines are no longer asserted. Normally, this will disconnect the line.

*cfgetispeed()* returns the input baud rate stored in the *termios* structure.

*cfsetispeed()* sets the input baud rate stored in the *termios* structure to *speed*. If the input baud rate is set to zero, the input baud rate will be specified by the value of the output baud rate. Both *cfsetispeed()* and *cfsetospeed()* return a value of zero if successful and -1 to indicate an error. Attempts to set unsupported baud rates are ignored, and it is implementation-defined whether an error is returned by any or all of *cfsetispeed()*, *cfsetospeed()*, or *tcsetattr()*. This refers both to changes to baud rates not supported by the hardware, and to changes setting the input and output baud rates to different values if the hardware does not support this.

## Standards Conformance

---

*cfgetispeed*, *cfgetospeed*, *cfsetispeed* and *cfsetospeed* are conformant with:

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

## chdir

change working directory

---

### Syntax

---

```
int chdir (path)
char *path;
```

### Description

---

*path* points to the path name of a directory. *chdir* causes the named directory to become the current working directory, the starting point for path searches for path names not beginning with /.

*chdir* will fail and the current working directory will be unchanged if one or more of the following is true:

- |             |                                                                                          |
|-------------|------------------------------------------------------------------------------------------|
| [ENOTDIR]   | A component of the path name is not a directory.                                         |
| [ENOENT]    | The named directory does not exist.                                                      |
| [EACCES]    | Search permission is denied for any component of the path name.                          |
| [EFAULT]    | <i>path</i> points outside the allocated address space of the process.                   |
| [EINTR]     | A signal was caught during the <i>chdir</i> system call.                                 |
| [ENOLINK]   | <i>path</i> points to a remote machine and the link to that machine is no longer active. |
| [EMULTIHOP] | Components of <i>path</i> require hopping to multiple remote machines.                   |

### See Also

---

chroot(S).

### Diagnostics

---

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.



## Standards Conformance

---

*chdir* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;

and NIST FIPS 151-1.

## **check\_basic\_data\_structures**

verify machine is suitable for security port

### **Syntax**

---

```
#include <sys/types.h>
#include <sys/security.h>
#include <sys/audit.h>
#include <prot.h>
```

```
void check_basic_data_structures ()
```

### **Description**

---

This routine is used to make sure the compiler and underlying architecture support constructs in the way expected by the secure OS port. Although every possible check cannot be made, enough checking is done to provide some degree of assurance that a machine/OS combination will not break assumptions built into the port.

Compiler checking verifies relationships between longs and shorts, chars and shorts, ... The padding in structures and arrangement of fields within structures is also checked.

Architectural checking identifies basic data units on the machine. The size of shorts and longs, as well as the default size of an int is tested, as well as other relationships.

### **Notes**

---

It is intended that this routine is to be expanded to support new architectures as they are tested with the port. This routine should catch problems, if not merely identify possible areas for consideration, before the port is operational. In security ports, any such guards are useful so that a buggy system can be avoided before it causes security breaches.

### **Value Added**

---

*check\_basic\_data\_structures* is an extension of AT&T System V provided by the Santa Cruz Operation.

## chmod

---

change mode of file

### Syntax

---

```
int chmod (path, mode)
char *path;
int mode;
```

### Description

---

The *path* argument points to a path name naming a file. The *chmod* system call sets the access permission portion of the named file's mode according to the bit pattern contained in *mode*.

Access permission bits are interpreted as follows:

04000

Set user ID on execution.

020#0

Set group ID on execution if # is 7, 5, 3, or 1

Enable mandatory file/record locking if # is 6, 4, 2, or 0

01000

Save text image after execution.

00400

Read by owner.

00200

Write by owner.

00100

Execute (search if a directory) by owner.

00070

Read, write, execute (search) by group.

00007

Read, write, execute (search) by others.

The effective user ID of the process must match the owner of the file or be super-user to change the mode of a file.

If the effective user ID of the process is not super-user and the file is not a directory, mode bit 01000 (save text image on execution) is cleared.



If the effective user ID of the process is not super-user and the effective group ID of the process does not match the group ID of the file, mode bit 02000 (set group ID on execution) is cleared.

If a 410 executable file has the sticky bit (mode bit 01000) set, the operating system will not delete the program text from the swap area when the last user process terminates. If a 413 executable file has the sticky bit set, the operating system will not delete the program text from memory when the last user process terminates. In either case, if the sticky bit is set, the text will already be available (either in a swap area or in memory) when the next user of the file executes it, thus making execution faster.

Overall, if a directory is writable and has the sticky bit set, files within that directory can only be removed if one or more of the following is true (see *unlink(S)*):

- the user owns the file
- the user owns the directory
- the file is writable to the user
- the user is the super-user

If the mode bit 02000 (set group ID on execution) is set and the mode bit 00010 (execute or search by group) is not set, mandatory file/record locking will exist on a regular file. This may effect future calls to *open(S)*, *creat(S)*, *read(S)*, and *write(S)* on this file.

*chmod* will fail and the file mode will be unchanged if one or more of the following is true:

|           |                                                                                                         |
|-----------|---------------------------------------------------------------------------------------------------------|
| [ENOTDIR] | A component of the path prefix is not a directory.                                                      |
| [ENOENT]  | The named file does not exist.                                                                          |
| [EACCES]  | Search permission is denied on a component of the path prefix.                                          |
| [EPERM]   | The effective user ID does not match the owner of the file and the effective user ID is not super-user. |
| [EROFS]   | The named file resides on a read-only file system.                                                      |
| [EFAULT]  | <i>path</i> points outside the allocated address space of the process.                                  |
| [EINTR]   | A signal was caught during the <i>chmod</i> system call.                                                |

- [ENOLINK] *path* points to a remote machine and the link to that machine is no longer active.
- [EMULTIHOP] Components of *path* require hopping to multiple remote machines.

## See Also

---

chown(S), creat(S), fcntl(S), mknod(S), open(S), read(S), write(S), chmod(C).

## Diagnostics

---

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## Standards Conformance

---

*chmod* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

## chown

change owner and group of a file

### Syntax

```
int chown (path, owner, group)
char *path;
int owner, group;
```

### Description

*path* points to a path name naming a file. The owner ID and group ID of the named file are set to the numeric values contained in *owner* and *group* respectively.

Only processes with effective user ID equal to the file owner or super-user may change the ownership of a file.

If *chown* is invoked by other than the super-user, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively, will be cleared.

*chown* will fail and the owner and group of the named file will remain unchanged if one or more of the following is true:

- |             |                                                                                                         |
|-------------|---------------------------------------------------------------------------------------------------------|
| [ENOTDIR]   | A component of the path prefix is not a directory.                                                      |
| [ENOENT]    | The named file does not exist.                                                                          |
| [EACCES]    | Search permission is denied on a component of the path prefix.                                          |
| [EPERM]     | The effective user ID does not match the owner of the file and the effective user ID is not super-user. |
| [EROFS]     | The named file resides on a read-only file system.                                                      |
| [EFAULT]    | <i>path</i> points outside the allocated address space of the process.                                  |
| [EINTR]     | A signal was caught during the <i>chown</i> system call.                                                |
| [ENOLINK]   | <i>path</i> points to a remote machine and the link to that machine is no longer active.                |
| [EMULTIHOP] | Components of <i>path</i> require hopping to multiple remote machines.                                  |



## See Also

---

`chmod(S)`, `chown(C)`.

## Diagnostics

---

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## Standards Conformance

---

*chown* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

# chroot

change root directory

## Syntax

```
int chroot (path)
char *path;
```

## Description

The *path* argument points to a path name naming a directory. The *chroot* system call causes the named directory to become the root directory, the starting point for path searches for path names beginning with /. The user's working directory is unaffected by the *chroot* system call.

The effective user ID of the process must be super-user to change the root directory.

The .. entry in the root directory is interpreted to mean the root directory itself. Thus, .. cannot be used to access files outside the subtree rooted at the root directory.

The *chroot* system call will fail and the root directory will remain unchanged if one or more of the following is true:

|             |                                                                                                       |
|-------------|-------------------------------------------------------------------------------------------------------|
| [ENOTDIR]   | Any component of the path name is not a directory.                                                    |
| [ENOENT]    | The named directory does not exist.                                                                   |
| [EPERM]     | The effective user ID is not super-user.                                                              |
| [EFAULT]    | The <i>path</i> argument points outside the allocated address space of the process.                   |
| [EINTR]     | A signal was caught during the <i>chroot</i> system call.                                             |
| [ENOLINK]   | The <i>path</i> argument points to a remote machine and the link to that machine is no longer active. |
| [EMULTIHOP] | Components of <i>path</i> require hopping to multiple remote machines.                                |

## See Also

chdir(S).

## Diagnostics

---

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## Standards Conformance

---

*chroot* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



## chsize

---

changes the size of a file.

### Syntax

---

```
int chsize (fildes, size)
int fildes;
long size;
```

### Description

---

*fildes* is a file descriptor obtained from a *creat*, *open*, *dup*, *fcntl*, or *pipe* system call. *chsize* changes the size of the file associated with the file descriptor *fildes* to be exactly *size* bytes in length. The routine either truncates the file, or pads it with an appropriate number of bytes. If *size* is less than the initial size of the file, then all allocated disk blocks between *size* and the initial file size are freed.

The maximum file size as set by *ulimit* (S) is enforced when *chsize* is called, rather than on subsequent writes. Thus *chsize* fails, and the file size remains unchanged if the new changed file size would exceed the *ulimit*.

### Return Value

---

Upon successful completion, a value of 0 is returned. Otherwise, the value -1 is returned and *errno* is set to indicate the error.

### See Also

---

*creat*(S), *dup*(S), *lseek*(S), *open*(S), *pipe*(S), *ulimit*(S)

### Notes

---

In general if *chsize* is used to expand the size of a file, when data is written to the end of the file, intervening blocks are filled with zeros. In a few rare cases, reducing the file size may not remove the data beyond the new end-of-file. This routine must be linked with the linker option *-lx*.

### Value Added

---

*chsize* is an extension of AT&T System V provided by the Santa Cruz Operation.

## clock

---

report CPU time used

### Syntax

---

`long clock ( )`

### Description

---

The *clock* function returns the amount of CPU time (in microseconds) used since the first call to *clock*. The time reported is the sum of the user and system times of the calling process and its terminated child processes for which it has executed *wait(S)*, *pclose(S)*, or *system(S)*.

The resolution of the clock is 10 milliseconds.

### See Also

---

`times(S)`, `wait(S)`, `popen(S)`, `system(S)`.

### Notes

---

The value returned by *clock* is defined in microseconds for compatibility with systems that have CPU clocks with much higher resolution. Because of this, the value returned will wrap around after accumulating only 2147 seconds of CPU time (about 36 minutes).

### Standards Conformance

---

*clock* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
and ANSI X3.159-198X C Language Draft Standard, May 13, 1988.

## close

---

close a file descriptor

### Syntax

---

```
int close (fildes)
int fildes;
```

### Description

---

The *fildes* argument is a file descriptor obtained from a *creat*, *open*, *dup*, *fcntl*, or *pipe* system call. The *close* system call closes the file descriptor indicated by *fildes*. All outstanding record locks owned by the process (on the file indicated by *fildes*) are removed.

If a STREAMS (see *intro*(S)) file is closed, and the calling process had previously registered to receive a SIGPOLL signal (see *signal*(S) and *sigset*(S)) for events associated with that file, the calling process will be unregistered for events associated with the file. The last *close* for a *stream* causes the *stream* associated with *fildes* to be dismantled. If O\_NDELAY is not set and there have been no signals posted for the *stream*, *close* waits up to 15 seconds, for each module and driver, for any output to drain before dismantling the *stream*. If the O\_NDELAY flag is set or if there are any pending signals, *close* does not wait for output to drain and dismantles the *stream* immediately.

The named file is closed unless one or more of the following is true:

- |           |                                                                                        |
|-----------|----------------------------------------------------------------------------------------|
| [EBADF]   | The <i>fildes</i> argument is not a valid open file descriptor.                        |
| [EINTR]   | A signal was caught during the <i>close</i> system call.                               |
| [ENOLINK] | <i>fildes</i> is on a remote machine and the link to that machine is no longer active. |

### See Also

---

*creat*(S), *dup*(S), *exec*(S), *fcntl*(S), *intro*(S), *open*(S), *pipe*(S), *signal*(S), *sigset*(S).

### Diagnostics

---

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.



## **Standards Conformance**

---

*close* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;  
and NIST FIPS 151-1.

# **strxfrm, strnxfm, strcoll, strncoll**

handles collation of strings

## **INTERNATIONAL VERSION**

### **Syntax**

```
strxfrm(to, from, maxsize)
char *to, *from;
int maxsize;
```

```
strnxfm(to, from, maxsize, n)
char *to, *from;
int maxsize, n;
```

```
strcoll(str1, str2)
char *str1, *str2;
```

```
strncoll(str1, str2, n)
char *str1, *str2;
int n;
```

### **Description**

These functions operate on null-terminated strings.

The *strxfrm* routine transforms the string *from* according to the collation environment of the current locale. The result is placed in the character array *to*. The maximum size of the resulting string is the value contained in the *maxsize* variable. The result can then be compared with another transformed string to establish the collating order of the two original strings. The *strnxfm* routine transforms at most *n* characters in the *from* string, where *n* is defined after the following collation rules have been applied:

- 1->2 character transformations : for example, the german '337' counts as two characters.
- 2->1 character transformations : for example, the spanish 'ch' counts as a single character.
- 1->1 character transformations : for example, 'a', 'b', etc. count as single characters.
- 1->0 character transformations : for example, the hyphen, '-', is not counted.

Both functions return the size of the array required to hold the transformed string. If the return value is greater than *maxsize* then the contents of *to* is undefined.

The *strcoll* function is used to collate the two strings *str1* and *str2* according to the collation environment of the current locale. The returned value is equal to, less than or greater than 0, according to whether *str1* is equal to, less than or greater than *str2*. *strncoll* collates the two strings until the *n*th character in *str1* is reached. The *n*th character is defined in the same way as *strnxfm*.

## See Also

---

coltbl(M), nl\_strcmp(S), string(S)

## Examples

---

It is assumed that the value of *maxsize* is large enough to contain the transformed string in the *strxfm* and the *strnxfm* examples.

Example 1:

This is an example of the *strxfm* routine:

```
char *to1, *to2;
int ret;

strxfm(to1, "Stra337e", maxsize);
strxfm(to2, "Strasse", maxsize);
ret= strcmp(to1, to2)
```

*ret* will contain 0 as "Stra337e" and "Strasse" collate as equal.

Example 2:

This is an example of the *strcoll* routine:

```
int ret;

ret = strcoll("Stra337e", "Strasse");
```

*ret* will contain 0 as "Stra337e" and "Strasse" collate as equal.



Example 3:

This is an example of the *strnxfrm* routine:

```
char *to;
```

```
strnxfrm("Stra337e", maxsize , 6);
```

This will transform only the "Stra337" portion of the string, as 337 counts as 2 characters.

Example 4:

This is an example of the *strncoll* routine:

```
strncoll("Stra337e", "Bahn", 6);
```

This will compare only the "Stra337" portion of the string, as 337 counts as two characters.

## **Standards Conformance**

---

*strcoll* and *strxfrm* are conformant with:

ANSI X3.159-198X C Language Draft Standard, May 13, 1988.

## **conv: toupper, tolower, \_toupper, \_tolower, toascii, todigit, toint**

---

routines used to translate characters

### **Syntax**

---

```
#include <ctype.h>
```

```
int toupper(c)
int c;
```

```
int tolower(c)
int c;
```

```
int _toupper(c)
int c;
```

```
int _tolower(c)
int c;
```

```
int toascii(c)
int c;
```

```
int todigit(i)
int i;
```

```
int toint(c)
int c;
```

### **Description**

---

The functions *toupper* and *tolower* convert the argument *c* to a letter of the opposite case. Arguments may be the integers -1 to 255 (the same values returned by *get(S)*). If the argument of *toupper* represents a lowercase letter, the result is the corresponding uppercase letter. If the argument of *tolower* represents an uppercase letter, the result is the corresponding lowercase letter. All other arguments are returned unchanged.

The effect of the macros *\_toupper* and *\_tolower* is the same as *toupper* and *tolower*. The advantage of *\_toupper* and *\_tolower* is that they have restricted argument values and are faster. The *\_toupper* macro takes a lowercase letter as its argument. The result is the corresponding uppercase letter. The *\_tolower* macro takes an uppercase letter as its argument. The result is the corresponding lowercase letter. All other arguments cause unpredictable results.

The macro *toascii* yields its argument with all bits cleared that are not part of a standard ASCII character; it is intended for compatibility with other systems.

The macro *todigit* returns the digit character corresponding to its integer argument. The argument must be in the range 0-9, otherwise the behavior is undefined.

The macro *toint* returns the integer corresponding to the digit character supplied as its argument. The argument must be one of 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, otherwise the behavior is undefined.

## See Also

---

ctype(S)

## Notes

---

The routines *toupper*, *tolower* and *toascii* are implemented as macros, and on other systems, *toupper* and *tolower* may be implemented as macros. On this system, all *conv(S)* routines evaluate their arguments once and once only. It is, however, good practice to ensure that macros are not called with arguments that have side effects which differ if they are evaluated more than once. An example of such a macro is:

```
toupper(*p++).
```

Avoiding such argument calls helps to keep code portable.

The functions *toupper* and *tolower* are library functions. Their arguments are converted to integers. For this reason, care should be taken that character arguments are supplied as unsigned characters to avoid problems with sign-extension of 8-bit character values.

The *todigit* and *toint* macros are extensions, and may not be present on other systems.



## Standards Conformance

---

*toascii* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

*tolower* and *toupper* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
ANSI X3.159-198X C Language Draft Standard, May 13,  
1988;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;  
and NIST FIPS 151-1.

## creat

---

create a new file or rewrite an existing one

### Syntax

---

```
#include <sys/types.h>
#include <sys/stat.h>
```

```
int creat (path, mode)
char *path;
int mode;
```

### Description

---

The *creat* system call creates a new ordinary file or prepares to rewrite an existing file named by the path name pointed to by *path*.

If the file exists, the length is truncated to 0 and the mode and owner are unchanged. Otherwise, the file's owner ID is set to the effective user ID of the process; the group ID of the process is set to the effective group ID of the process; and the low-order 12 bits of the file mode are set to the value of *mode* modified as follows:

All bits set in the process's file mode creation mask are cleared (see *umask*(S)).

The "save text image after execution bit" of the mode is cleared (see *chmod*(S)).

Upon successful completion, a write-only file descriptor is returned and the file is open for writing, even if the mode does not permit writing. The file pointer is set to the beginning of the file. The file descriptor is set to remain open across *exec* system calls (see *fcntl*(S)). No process may have more than 20 files open simultaneously. A new file may be created with a mode that forbids writing.

Symbolic constants defining the access permission bits are specified in the *<sys/stat.h>* header file and should be used to construct *mode* (see *chmod*(S)).

The call **creat(path, mode)** is equivalent to the following (see *open*(S)):

```
open(path, O_WRONLY | O_CREAT | O_TRUNC, mode)
```

The *creat* system call fails if one or more of the following is true:

|             |                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------|
| [ENOTDIR]   | A component of the path prefix is not a directory.                                                                                |
| [ENOENT]    | A component of the path prefix does not exist.                                                                                    |
| [EACCES]    | Search permission is denied on a component of the path prefix.                                                                    |
| [ENOENT]    | The path name is null.                                                                                                            |
| [EACCES]    | The file does not exist and the directory in which the file is to be created does not permit writing.                             |
| [EROFS]     | The named file resides or would reside on a read-only file system.                                                                |
| [ETXTBSY]   | The file is a pure procedure (shared text) file that is being executed.                                                           |
| [EACCES]    | The file exists and write permission is denied.                                                                                   |
| [EISDIR]    | The named file is an existing directory.                                                                                          |
| [EMFILE]    | NOFiles file descriptors are currently open.                                                                                      |
| [EFAULT]    | The <i>path</i> argument points outside the allocated address space of the process.                                               |
| [ENFILE]    | The system file table is full.                                                                                                    |
| [EAGAIN]    | The file exists, mandatory file/record locking is set, and there are outstanding record locks on the file (see <i>chmod(S)</i> ). |
| [EINTR]     | A signal was caught during the <i>creat</i> system call.                                                                          |
| [ENOLINK]   | <i>path</i> points to a remote machine and the link to that machine is no longer active.                                          |
| [EMULTIHOP] | Components of <i>path</i> require hopping to multiple remote machines.                                                            |
| [ENOSPC]    | The file system is out of inodes.                                                                                                 |

## See Also

*chmod(S)*, *close(S)*, *dup(S)*, *fcntl(S)*, *lseek(S)*, *open(S)*, *read(S)*, *umask(S)*, *write(S)*.



## Diagnostics

---

Upon successful completion, a non-negative integer, namely the file descriptor, is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## Standards Conformance

---

*creat* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;  
and NIST FIPS 151-1.

## creatsem

---

creates an instance of a binary semaphore

### Syntax

---

```
int creatsem(sem_name,mode)
char *sem_name;
int mode;
```

### Description

---

*creatsem* defines a binary semaphore named by *sem\_name* to be used by *waitsem*(S) and *sigsem*(S) to manage mutually exclusive access to a resource, shared variable, or critical section of a program. *creatsem* returns a unique semaphore number, *sem\_num*, which may then be used as the parameter in *waitsem* and *sigsem* calls. Semaphores are special files of 0 length. The filename space is used to provide unique identifiers for semaphores. *mode* sets the accessibility of the semaphore using the same format as file access bits. Access to a semaphore is granted only on the basis of the read access bit; the write and execute bits are ignored.

A semaphore can be operated on only by a synchronizing primitive, such as *waitsem* or *sigsem*, by *creatsem* which initializes it to some value, or by *opensem* which opens the semaphore for use by a process. Synchronizing primitives are guaranteed to be executed without interruption once started. These primitives are used by associating a semaphore with each resource (including critical code sections) to be protected.

The process controlling the semaphore should issue:

```
sem_num = creatsem("semaphore", mode);
```

to create, initialize, and open the semaphore for that process. All other processes using the semaphore should issue:

```
sem_num = opensem("semaphore");
```

to access the semaphore's identification value. Note that a process cannot open and use a semaphore that has not been initialized by a call to *creatsem*, nor should a process open a semaphore more than once in one period of execution. Both the creating and opening processes use *waitsem* and *sigsem* to use the semaphore *sem\_num*.

## Compatibility

---

*creatsem* can only be used to define UNIX version 3.0 type semaphores, not UNIX System V type semaphores.

## See Also

---

*opensem*(S), *waitsem*(S), *sigsem*(S)

## Diagnostics

---

*creatsem* returns the value -1 if an error occurs. If the semaphore named by *sem\_name* is already open for use by other processes, *errno* is set to EEXIST. If the file specified exists but is not a semaphore type, *errno* is set to ENOTNAM. If the semaphore has not been initialized by a call to *creatsem*, *errno* is set to ENAVAIL.

## Notes

---

After a *creatsem* you must do a *waitsem* to gain control of a given resource.

This feature is a XENIX specific enhancement and may not be present in all UNIX implementations. This function must be linked with the linker option *-lx*.

## Value Added

---

*creatsem* is an extension of AT&T System V provided by the Santa Cruz Operation.



## crypt

---

password and file encryption functions

### Syntax

---

```
cc [flag ...] file ...-lcrypt
```

```
char *crypt (key, salt)
char *key, *salt;
```

```
void setkey (key)
char *key;
```

```
void encrypt (block, flag)
char *block;
int flag;
```

```
char *des_crypt (key, salt)
char *key, *salt;
```

```
void des_setkey (key)
char *key;
```

```
void des_encrypt (block, flag)
char *block;
int flag;
```

```
int run_setkey (p, key)
int p[2];
char *key;
```

```
int run_crypt (offset, buffer, count, p)
long offset;
char *buffer;
unsigned int count;
int p[2];
```

```
int crypt_close(p)
int p[2];
```

### Description

---

*des\_crypt* is the password encryption function. It is based on a one-way hashing encryption algorithm with variations intended (among other things) to frustrate use of hardware implementations of a key search.

*key* is a user's typed password. *salt* is a two-character string chosen from the set [a-zA-Z0-9./]. This string is used to perturb the hashing algorithm in one of 4096 different ways, after which the password is used as the key to encrypt repeatedly a constant string. The returned value points to the encrypted password. The first two characters are the salt itself.

The *des\_setkey* and *des\_encrypt* entries provide (rather primitive) access to the actual hashing algorithm. The argument of *des\_setkey* is a character array of length 64 containing only the characters with numerical value 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored; this gives a 56-bit key which is set into the machine. This is the key that will be used with the hashing algorithm to encrypt the string *block* with the function *des\_encrypt*.

The argument to the *des\_encrypt* entry is a character array of length 64 containing only the characters with numerical value 0 and 1. The argument array is modified in place to a similar array representing the bits of the argument after having been subjected to the hashing algorithm using the key set by *des\_setkey*. If *edflag* is zero, the argument is encrypted; if non-zero, it is decrypted.

Note that decryption is not provided in the international version of *crypt(S)*. If decryption is attempted with the international version of *des\_encrypt*, an error message is printed.

*crypt*, *setkey*, and *encrypt* are front-end routines that invoke *des\_crypt*, *des\_setkey*, and *des\_encrypt* respectively.

The routines *run\_setkey* and *run\_crypt* are designed for use by applications that need cryptographic capabilities (such as *ed(C)* and *vi(C)*) that must be compatible with the *crypt(C)* user-level utility. *run\_setkey* establishes a two-way pipe connection with *crypt(C)*, using *key* as the password argument. *run\_crypt* takes a block of characters and transforms the cleartext or ciphertext using *crypt(C)*. *Offset* is the relative byte position from the beginning of the file that the block of text provided in *block* is coming from. *count* is the number of characters in *block*, and *connection* is an array containing indices to a table of input and output file streams. When encryption is finished, *crypt\_close* is used to terminate the connection with *crypt(C)*.

*run\_setkey* returns -1 if a connection with *crypt(C)* cannot be established. This will occur on international versions of the Operating System where *crypt(C)* is not available. If a null key is passed to *run\_setkey*, 0 is returned. Otherwise, 1 is returned. *run\_crypt* returns -1 if it cannot write output or read input from the pipe attached to *crypt*. Otherwise it returns 0.

## See Also

---

`crypt(S)`, `getpass(S)`, `passwd(F)`, `crypt(C)`, `login(M)`, `passwd(C)`

## Diagnostics

---

In the international version of *crypt(S)*, a flag argument of 1 to *des\_encrypt* is not accepted, and an error message is printed.

## Note

---

The return value in *crypt* points to static data that are overwritten by each call.

## Standards Conformance

---

*crypt* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



## ctermid

---

generate file name for terminal

### Syntax

---

```
#include <stdio.h>
char *ctermid (s)
char *s;
```

### Description

---

The *ctermid* function generates the path name of the controlling terminal for the current process and stores it in a string.

If *s* is a NULL pointer, the string is stored in an internal static area, the contents of which are overwritten at the next call to *ctermid*, and the address of which is returned. Otherwise, *s* is assumed to point to a character array of at least **L\_ctermid** elements; the path name is placed in this array, and the value of *s* is returned. The constant **L\_ctermid** is defined in the *<stdio.h>* header file.

### See Also

---

*ttyname*(S)

### Notes

---

The difference between *ctermid* and *ttyname*(S) is that *ttyname* must be handed a file descriptor and returns the actual name of the terminal associated with that file descriptor, while *ctermid* returns a string (*/dev/tty*) that will refer to the terminal if used as a file name. Thus *ttyname* is useful only if the process already has at least one file open to a terminal.

### Standards Conformance

---

*ctermid* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

# **ctime, localtime, gmtime, asctime, cftime, ascftime, strftime, tzset**

---

convert date and time to string

## **Syntax**

---

```
#include <sys/types.h>
#include <time.h>

char *ctime (clock)
time_t *clock;

struct tm *localtime (clock)
time_t *clock;

struct tm *gmtime (clock)
time_t *clock;

char *asctime (tm)
const struct tm *tm;

int cftime(buf, fmt, clock)
char *buf, *fmt;
time_t *clock;

int ascftime (buf, fmt, tm)
char *buf, *fmt;
struct tm *tm;

size_t strftime(s, maxsize, format, timeptr)
char *s;
size_t maxsize;
const *format;
const struct tm *timeptr;

void tzset ()

extern long timezone, altzone;

extern int daylight;

extern char *tzname[2];

#include <time.h>
```

## Description

---

*ctime*,  
*localtime*,  
 and  
*gmtime*  
 accept arguments of type  
**time\_t**  
 (declared in <sys/types.h>),  
 pointed to by  
*clock*,  
 representing the time in seconds since  
 00:00:00 Greenwich Mean Time (GMT), January 1, 1970.  
*ctime* returns a pointer to a 26-character string in the following form.  
 All the fields have constant width.

Fri Sep 13 00:00:00 1986\n\0

*localtime*  
 and  
*gmtime*  
 return pointers to “tm” structures, described below.  
*localtime*  
 corrects for the main time zone and possible alternate (“Daylight Savings”) time zone;  
*gmtime*  
 converts directly to GMT, which is the time the system uses.

*asctime*  
 converts a “tm” structure to a 26-character string,  
 as shown in the above example, and returns a pointer to the string.

Declarations of all the functions and externals, and the “tm” structure, are in the <time.h> header file.

The structure declaration is:

```
struct tm {
 int tm_sec; /* seconds after the minute - [0, 59] */
 int tm_min; /* minutes after the hour - [0, 59] */
 int tm_hour; /* hour since midnight - [0, 23] */
 int tm_mday; /* day of the month - [1, 31] */
 int tm_mon; /* months since January - [0, 11] */
 int tm_year; /* years since 1900 */
 int tm_wday; /* days since Sunday - [0, 6] */
 int tm_yday; /* days since January 1 - [0, 365] */
 int tm_isdst; /* flag for daylight savings time */
};
```

*tm\_isdst* is non-zero if the alternate time zone is in effect.



*cftime* and *ascftime* provide the capabilities of *ctime* and *asctime*, respectively, as well as additional ones. *cftime* takes an integer of type *time\_t* pointed to by *clock* and converts it to a character string. *ascftime* takes a pointer to a "tm" structure and converts it to a character string. In both functions, the characters are placed into the array pointed to by *buf* (plus a terminating \0) and the value returned is the number of such characters (not counting the terminating \0). *fmt* controls the format of the resulting string.

*fmt* is a character string that consists of field descriptors and text characters, reminiscent of *printf*(S). Each field descriptor consists of a % character followed by another character which specifies the replacement for the field descriptor. All other characters are copied from *fmt* into the result. The following field descriptors are supported:

|     |                                                                  |
|-----|------------------------------------------------------------------|
| % % | same as %                                                        |
| %a  | abbreviated weekday name                                         |
| %A  | full weekday name                                                |
| %b  | abbreviated month name                                           |
| %B  | full month name                                                  |
| %d  | day of month ( 01 - 31 )                                         |
| %D  | date as %m/%d/%y                                                 |
| %e  | day of month (1-31; single digits are preceded by a blank)       |
| %h  | abbreviated month name                                           |
| %H  | hour ( 00 - 23 )                                                 |
| %I  | hour ( 00 - 12 )                                                 |
| %j  | day number of year ( 001 - 366 )                                 |
| %m  | month number ( 01 - 12 )                                         |
| %M  | minute ( 00 - 59 )                                               |
| %n  | same as \n                                                       |
| %p  | ante meridian or post meridian                                   |
| %r  | time as %I:%M:%S %p                                              |
| %R  | time as %H:%M                                                    |
| %S  | seconds ( 00 - 59 )                                              |
| %t  | insert a tab                                                     |
| %T  | time as %H:%M:%S                                                 |
| %U  | week number of year ( 01 - 52 ), Sunday is the first day of week |
| %w  | weekday number ( Sunday = 0 )                                    |
| %W  | week number of year ( 01 - 52 ), Monday is the first day of week |

|    |                                 |
|----|---------------------------------|
| %x | Local specific date format      |
| %X | Local specific time format      |
| %y | year within century ( 00 - 99 ) |
| %Y | year as cyy ( e.g. 1986)        |
| %Z | time zone name                  |

The difference between %U and %W lies in which day is counted as the first of the week. Week number 01 is the first week with four or more January days in it.

The example below shows what the values in the "tm" structure would look like for Thursday, August 28, 1986, at 12:44:36 in New Jersey.

```
asctime (buf, "%A %m %d %j", tm)
```

This example would result in the buffer containing "Thursday Aug 28 240".

If *fmt* is (char \*)0, the value of the environment variable **CFTIME** is used. If **CFTIME** is undefined or empty, a default format is used. The default format string is taken from the file that contains the date and time strings associated with the then current language (see below for details on changing the current language and *cftime*(F) for a description of the structure of these files).

The user can request that the output of *cftime* and *asctime* be in a specific language by setting the environment variable **LANGUAGE** to the desired language. If **LANGUAGE** is empty, unset or set to an unsupported language, the last language requested will be used (the default is the **usa-english** strings).

The *strftime* function places its output (according to the format string *format* and the time values contained in the structure pointed to by *timeptr*), followed by the null character (\0) in consecutive bytes starting at *s*. The maximum length of the output string is specified by *max-size*, the maximum number of characters including the terminating null character.

The *strftime* function converts and formats the time values contained in the structure pointed to by *timeptr* under control of the format string *format*. The string *format* is a character string can contain two types of object. These are: plain characters, which are simply copied to the output string, and directives.

Each directive is introduced by the percent character (%). The following directives are independent of the program's current locale (see *locale*(M)):



- %d** is replaced by the day of the month as a decimal number (01-31)
- %H** is replaced by the hour (24-hour clock) as a decimal number (00-23)
- %I** is replaced by the hour (12-hour clock) as a decimal number (01-12)
- %j** is replaced by the day of the year as a decimal number (001-366)
- %m** is replaced by the month as a decimal number (01-12)
- %M** is replaced by the minute as a decimal number (00-59)
- %S** is replaced by the second as a decimal number (00-59)
- %U** is replaced by the week number of the year (Sunday as the first day of the week) as a decimal number (00-52)
- %w** is replaced by the weekday as a decimal number [0(Sunday)-6]
- %W** is replaced by the week number of the year (Monday as the first day of the week) as a decimal number (00-52)
- %y** is replaced by the year without century as a decimal number (00-99)
- %Y** is replaced by the year with century as a decimal number
- %Z** is replaced by the timezone name, or by no characters if no timezone exists
- %%** is replaced by %

The strings used to replace the following directives are dependent on the program's current locale, and are defined using the utility *timtbl(M)*:

- %a** is replaced by the locale's abbreviated weekday name
- %A** is replaced by the locale's full weekday name
- %b** is replaced by the locale's abbreviated month name
- %B** is replaced by the locale's full month name
- %c** is replaced by the locale's appropriate date and time representation



- %p** is replaced by the locale's equivalent of AM or PM
- %x** is replaced by the locale's appropriate date representation
- %X** is replaced by the locale's appropriate time representation

The external **long** variable *timezone* contains the difference, in seconds, between GMT and the main time zone; the external **long** variable *altzone* contains the difference, in seconds, between GMT and the alternate time zone; both, *timezone* and *altzone* default to 0 (GMT). The external variable *daylight* is non-zero if an alternate time zone exists. The time zone names are contained in the external variable *tzname*, which by default is set to

```
char *tzname[2] = { "GMT", " " };
```

The functions know about the peculiarities of this conversion for various time periods for the U.S.A (specifically, the years 1974, 1975, and 1987). The functions will handle the new daylight savings time starting with the first Sunday in April, 1987, 1989.

*tzset* uses the contents of the environment variable **TZ** to override the value of the different external variables. The syntax of **TZ** can be described as follows:

```
TZ → zone
 / zone signed time
 / zone signed time zone
 / zone signed time zone dst
zone → letter letter letter
signed_time → sign time
 / time
time → hour
 / hour : minute
 / hour : minute :
dst → signed_time
 / signed time ; dst_date ,
 / ; dst_date , dst_date
dst_date → julian
 / julian / time
letter → a / A / b / B / ... / z / Z
hour → 00 / 01 / ... / 23
minute → 00 / 01 / ... / 59
second → 00 / 01 / ... / 59
julian → 001 / 002 / ... / 366
sign → - / +
```

*tzset* scans the contents of the environment variable and assigns the different fields to the respective variable. For example, the setting for New Jersey in 1986 could be

"EST5EDT4;117/2:00:00,299/2:00:00" .

or simply

EST5EDT

A southern hemisphere setting such as the Cook Islands could be

"KDT9:30KST10:00;64/5:00,303/20:00"

When the longer format is used, the variable must be surrounded by double quotes as shown. For more details, see *timezone(F)* and *environ(M)*. In the longer version of the New Jersey example of *TZ*, *tzname[0]* is EST, *timezone* will be set to 5\*60\*60, *tzname[1]* is EDT, *altzone* will be set to 4\*60\*60, the starting date of the alternate time zone is the 117th day at 2 AM, the ending date of the alternate time zone is the 299th day at 2 AM, and *daylight* will be set to non-zero. Starting and ending times are relative to the alternate time zone. If the alternate time zone start and end dates and the time are not provided, the days for the United States that year will be used and the time will be 2 AM. If the start and end dates are provided but the time is not provided, the time will be midnight. The effects of *tzset* are thus to change the values of the external variables *timezone*, *altzone*, *daylight*, and *tzname*. *tzset* is called by *localtime* and may also be called explicitly by the user.

Note that in most installations, *TZ* is set to the correct value by default when the user logs on, via the local */etc/profile* file (see *profile(M)*).

## Return Value

---

The *strftime* function returns the number of characters placed in the string *s* (not including the terminating null character), or zero if the length of the output string would exceed *maxsize*. In the latter case, the content of the output string *s* is undefined.

Zero is also returned by *strftime* if the function detects a situation which would lead to infinite recursion; for example, if a format string refers to itself.

## Files

---

*/lib/cftime* - directory that contains the language specific printable files

## See Also

---

*time(S)*, *timtbl(M)*, *getenv(S)*, *putenv(S)*, *printf(S)*, *cftime(F)*, *nl\_cxtime(S)* *profile(M)*, *timezone(F)*, *environ(M)*.

## Note

---

The return values for *ctime*, *localtime* and *gmtime* point to static data whose content is overwritten by each call.

Setting the time during the interval of change from *timezone* to *altzone* or vice versa can produce unpredictable results.

The system administrator must change the Julian start and end days annually if the full form of the TZ variable is specified.

## Standards Conformance

---

*asctime*, *ctime*, *gmtime* and *localtime* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

*strftime* is conformant with:

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

*tzset* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.



## ctype

---

### character handling routines

### Syntax

---

```
#include <ctype.h>
```

```
int isdigit (c);
int c;
```

```
...
```

```
tolower(c)
int c;
```

```
...
```

```
int setchrclass (chrclass)
char *chrclass;
```

### Description

---

The character classification macros listed below are defined by the `<ctype.h>` header file. They each return nonzero for true, zero for false. *isascii* is defined on all integer values; the rest are defined on valid members of the character set and on the single value EOF (see *stdio(S)*) (guaranteed not to be a character set member).

*isdigit*                tests for the digits 0 through 9.

*isxdigit*             tests for any character for which *isdigit* is true or for the letters *a* through *f* or *A* through *F*.

*islower*              tests for any lowercase letter as defined by the character set.

*isupper*              tests for any uppercase letter as defined by the character set.

*isalpha*             tests for any character for which *islower* or *isupper* is true and possibly any others as defined by the character set.

*isalnum*             tests for any character for which *isalpha* or *isdigit* is true.

|                |                                                                                                                                                      |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>isspace</i> | tests for a space, horizontal-tab, carriage return, newline, vertical-tab, or form-feed.                                                             |
| <i>isctrl</i>  | tests for "control characters" as defined by the character set.                                                                                      |
| <i>ispunct</i> | tests for any character other than the ones for which <i>isalnum</i> , <i>isctrl</i> , or <i>isspace</i> is true or space.                           |
| <i>isprint</i> | tests for a space or any character for which <i>isalnum</i> or <i>ispunct</i> is true or other "printing character" as defined by the character set. |
| <i>isgraph</i> | tests for any character for which <i>isprint</i> is true, except for space.                                                                          |
| <i>isascii</i> | tests for an ASCII character (a non-negative number less than 0200).                                                                                 |

The conversion functions and macros translate a character from lower-case (uppercase) to uppercase (lowercase).

|                       |                                                                                                                                                                                                                          |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>tolower</i>        | if the character is one for which <i>isupper</i> is true and there is a corresponding lowercase character, <i>tolower</i> returns the corresponding lowercase character. Otherwise, the character is returned unchanged. |
| <i>toupper</i>        | if the character is one for which <i>islower</i> is true and there is a corresponding uppercase character, <i>toupper</i> returns the corresponding uppercase character. Otherwise, the character is returned unchanged. |
| <i>toascii</i>        | turns off the bits that are not part of the ASCII character set.                                                                                                                                                         |
| <u><i>tolower</i></u> | returns the lowercase representation of a character for which <i>isupper</i> is true, otherwise undefined.                                                                                                               |
| <u><i>toupper</i></u> | returns the uppercase representation of a character for which <i>islower</i> is true, otherwise undefined.                                                                                                               |

The conversion macros have the same functionality of the functions on valid input, but the macros are faster because they do not do range checking.

All the character classification macros and the conversion functions and macros do a table lookup.

**setchrclass** initializes the table used by these functions and macros to a specific character classification set. **setchrclass** uses the value of its argument or the value of the environment variable **CHRCCLASS** as the name of the datafile containing the information for the desired character set. These datafiles are searched for in the special directory **/lib/chrclass**.

If *chrclass* is (char \*)0, the value of the environment variable **CHRCCLASS** is used. If **CHRCCLASS** is not set or is undefined, the table retains its current value, which at initialization time is **ascii**.

## Files

---

**/lib/chrclass** - directory containing the datafiles for **setchrclass**

## See Also

---

**chrtbl(M)**, **stdio(S)**, **ascii(M)**, **environ(M)**

## Diagnostics

---

If the argument to any of the character handling macros is not in the domain of the function, the result is undefined.

If **setchrclass** does not successfully fill the table, the table will not change (initially "ascii") and -1 is returned. If everything works, **setchrclass** returns 0.

## Warning

---

If a character variable or constant is passed to these functions or macros, undefined results may occur on machines which sign-extend characters by default.

## Standards Conformance

---

*isalnum*, *isalpha*, *isctrl*, *isdigit*, *isgraph*, *islower*, *isprint*, *ispunct*, *isspace*, *isupper*, *isxdigit*, *tolower* and *toupper* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.



CTYPE (S)

CTYPE (S)

*\_tolower* and *\_toupper* are conformant with:  
AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## curSES

---

terminal screen handling and optimization package

### Syntax

---

The *curSES* manual page is organized as follows:

In Syntax

- compiling information
- summary of parameters used by *curSES* routines
- alphabetical list of *curSES* routines, showing their parameters

In Description:

- An overview of how *curSES* routines should be used

In ROUTINES, each *curSES* routine is described under the appropriate heading:

- Overall Screen Manipulation
- Window and Pad Manipulation
- Output
- Input
- Output Options Setting
- Input Options Setting
- Environment Queries
- Color Manipulation
- Soft Labels
- Low-level Curses Access
- Terminfo-Level Manipulations
- Termcap Emulation
- Miscellaneous
- Use of **curscr**

Then come sections on:

- ATTRIBUTES
- COLORS
- FUNCTION KEYS
- LINE GRAPHICS

### Syntax

---

There are two different *curSES* libraries for both COFF and XENIX binaries, "termcap" *curSES* and "terminfo" *curSES*. Each version of *curSES* uses a different terminal capabilities database, has different functionality, and has a corresponding header file. Terminfo *curSES* is the default at installation and can be selected by inserting the following line in your source code:

```
#include < curses.h>
```

and using the following flag on the *cc*(CP) command line:

```
-lcurses [libraries . .]
```

**-lcurses** selects the default *curses* library, */usr/lib/libcurses.a* (COFF) or */lib/386/Slibcurses.a* (x.out). The libraries are links to the terminfo *curses* libraries */usr/lib/libtinfo.a* (COFF) or */lib/386/Slibtinfo.a* (X.OUT). */usr/include/curses.h* defines *M\_TERMINFO*, which automatically includes */usr/include/tinfo.h*.

This default may be changed to use termcap *curses*. Modify the */usr/include/curses.h* file to define *M\_TERMCAP* instead of *M\_TERMINFO* to change the default. If you make this change, the compiler will include */usr/include/tcap.h* and link the default *curses* library to the termcap *curses* library. For COFF libraries, link the file */usr/lib/libcurses.a* to */usr/lib/libtcap.a*. For x.out libraries, link the file */lib/386/libcurses.a* to */lib/386/libtcap.a*. Termcap *curses* requires an extra library and may now be used as follows. Include the following line in your source code:

```
#include < curses.h>
```

and compile with the following flag:

```
-lcurses -lterm lib [libraries...]
```

Note that for non-386 x.out binaries, a number of links must be changed, one for each possible memory model. Link */lib/[SMLC]libcurses.a* to */lib/[SMLC]libtcap.a*.

In either case, you can override the default and manually select either terminfo *curses* or termcap *curses*. For terminfo *curses*, include the following line in your source code:

```
#include < curses.h>
```

and compile with the following flags:

```
-DM_TERMINFO file -ltinfo [libraries...]
```

For termcap *curses*, include the following line in your source code:

```
#include < curses.h>
```

And compile with the following options:

```
-DM_TERMCAP file -ltcap -lterm lib [libraries...]
```



For the most part, termcap *cursets* is a subset of terminfo *cursets*. The following documentation serves for both libraries. Those routines marked with an asterisk "\*" are found in terminfo *cursets* only and are not available services under termcap *cursets*.

The parameters in the following list are not global variables, but rather this is a summary of the parameters used by the *cursets* library routines. All routines return the **int** values **ERR** or **OK** unless otherwise noted. Routines that return pointers always return **NULL** on error. (**ERR**, **OK**, and **NULL** are all defined in <cursets.h>.)

**bool** bf

**char** \*\*area, \*boolnames[ ], \*boolcodes[ ], \*boolfnames[ ], \*bp  
**char** \*cap, \*capname, codename[2], erasechar, \*filename, \*fmt  
**char** \*keyname, killchar, \*label, \*longname  
**char** \*name, \*numnames[ ], \*numcodes[ ], \*numfnames[ ]  
**char** \*slk\_label, \*str, \*strnames[ ], \*strcodes[ ], \*strfnames[ ]  
**char** \*term, \*tgetstr, \*tigetstr, \*tgoto, \*tparm, \*type

**chtype** attrs, ch, horch, vertch

**FILE** \*infd, \*outfd

**int** begin\_x, begin\_y, begline, bot, c, col, count

**int** dmaxcol, dmaxrow, dmincol, dminrow, \*erret, fildes

**int** (\*init( )), labfmt, labnum, line

**int** ms, ncols, new, newcol, newrow, nlines, numlines

**int** oldcol, oldrow, overlay

**int** p1, p2, p9, pmincol, pminrow, (\*putc( )), row

**int** smaxcol, smaxrow, smincol, sminrow, start

**int** tenths, top, visibility, x, y

**short** pair, f, b, color, r, g, b

**SCREEN** \*new, \*newterm, \*set\_term

**TERMINAL** \*cur\_term, \*nterm, \*oterm

**va\_list** varglist

**WINDOW** \*curscr, \*dstwin, \*initscr, \*newpad, \*newwin, \*orig

**WINDOW** \*pad, \*srcwin, \*stdscr, \*subpad, \*subwin, \*win

**addch**(ch)

**addkey**(str,bf)

**addstr**(str)

**attroff**(attrs)

**attron**(attrs)

**attrset**(attrs)

**\*baudrate**( )

**beep**( )

**box**(win, vertch, horch)

**\*can\_change\_color**( )

**cbreak**( ), **crmode**( )

**clear**( )

**clearok**(win, bf)

**clrtoeb**( )

**clrtoeol**( )

```

*color_content(color, &r, &g, &b)
*copywin(srcwin, dstwin, sminrow, smincol, dminrow, dmincol,
 dmaxrow, dmaxcol, overlay)
*curs_set(visibility)
curon() / termcap only */
curoff() / termcap only */
*def_prog_mode()
*def_shell_mode()
*del_curterm(oterm)
*delay_output(ms)
delch()
deleteln()
delwin(win)
*doupdate()
*draino(ms)
echo()
echochar(ch)
endwin()
erase()
erasechar()
*filter()
*flash()
*flushinp()
*garbagedlines(win, begline, numlines)
*getbegyx(win, y, x)
getorg(w, y, x) /* termcap only */
getch()
*getmaxyx(win, y, x)
getdim(w, y, x) /* termcap only */
getstr(str)
*getsyx(y, x)
getyx(win, y, x)
*halfdelay(tenths)
*has_colors()
*has_ic()
*has_il()
*idlok(win, bf)
inch()
*init_color(color, r, g, b)
*init_pair(pair, f, b)
initscr()
insch(ch)
insertln()
*intrflush(win, bf)
*isendwin()
is_standout()
*keyname(c)
keypad(win, bf)
*killchar()
leaveok(win, bf)
longname()
*meta(win, bf)

```

```

move(y, x)
mvaddch(y, x, ch)
mvaddstr(y, x, str)
mvcur(oldrow, oldcol, newrow, newcol)
mvdelch(y, x)
mvgetch(y, x)
mvgetstr(y, x, str)
mvinch(y, x)
mvinsch(y, x, ch)
mvprintw(y, x, fmt [, arg ...])
mvscanw(y, x, fmt [, arg ...])
mvwaddch(win, y, x, ch)
mvwaddstr(win, y, x, str)
mvwdelch(win, y, x)
mvwgetch(win, y, x)
mvwgetstr(win, y, x, str)
mvwin(win, y, x)
mvwinch(win, y, x)
mvwinsch(win, y, x, ch)
mvwprintw(win, y, x, fmt [, arg ...])
mvwscanw(win, y, x, fmt [, arg ...])
*napms(ms)
*newpad(nlines, ncols)
*newterm(type, outfd, infd)
newwin(nlines, ncols, begin_y, begin_x)
nl()
nocbreak(), nocrmode()
nodelay(win, bf)
noecho()
nonl()
noraw()
*notimeout(win, bf)
overlay(srcwin, dstwin)
overwrite(srcwin, dstwin)
*pair_content(pair, &f, &b)
*pechochar(pad, ch)
*pnoutrefresh(pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol)
*prefresh(pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol)
printw(fmt [, arg ...])
*putp(str)
raw()
refresh()
*reset_prog_mode()
reset_tty() /* termcap only */
*reset_shell_mode()
set_tty() /* termcap only */
resetty()
*restartterm(term, fildes, errret)
*ripline(line, init)
savetty()
scanw(fmt [, arg ...])
*scr_dump(filename)

```



```

dumpwin(w, file, margin) /* termcap only */
*scr_init(filename)
*scr_restore(filename)
scroll(win)
scrollok(win, bf)
*set_curterm(nterm)
*set_term(new)
*setscrreg(top, bot)
*setsyx(y, x)
*setupterm(term, fildes, errret)
*slk_clear()
*slk_init(fmt)
*slk_label(labnum)
*slk_noutrefresh()
*slk_refresh()
*slk_restore()
*slk_set(labnum, label, fmt)
*slk_touch()
standend()
standout()
*start_color()
*subpad(orig, nlines, ncols, begin_y, begin_x)
subwin(orig, nlines, ncols, begin_y, begin_x)
tgetent(bp, name)
tgetflag(codename)
tgetnum(codename)
tgetstr(codename, area)
tgoto(cap, col, row)
*tigetflag(capname)
*tigetnum(capname)
*tigetstr(capname)
*touchline(win, start, count)
touchwin(win)
tparm(str, p1, p2, ..., p9)
tputs(str, count, putc)
*traceoff()
*traceon()
*typeahead(fildes)
unctrl(c)
ungetch(c)
*vidattr(attrs)
*vidputs(attrs, putc)
*vwprintw(win, fmt, varglist)
*vwscanw(win, fmt, varglist)
waddch(win, ch)
waddstr(win, str)
wattroff(win, attrs)
wattron(win, attrs)
wattrset(win, attrs)
wclear(win)
wclrtoebot(win)
wclrtoeol(win)

```

```

wdelch(win)
wdeleteln(win)
wechochar(win, ch)
werase(win)
wgetch(win)
wgetstr(win, str)
winch(win)
winsch(win, ch)
winsertln(win)
wmove(win, y, x)
*wnoutrefresh(win)
wprintw(win, fmt [, arg ...])
wrefresh(win)
wscanw(win, fmt [, arg ...])
*wssetscrreg(win, top, bot)
wstandend(win)
wstandout(win)

```

## Description

---

The *curses* routines give the user a terminal-independent method of updating screens with reasonable optimization.

The file `<curses.h>` must be included at the beginning of programs that use any *curses* routines. In addition, the routine `initscr()` or `newterm()` must be called before any of the other routines that deal with windows and screens are used. (Three exceptions are noted where they apply.) The routine `endwin()` must be called before exiting. To get character-at-a-time input without echoing (most interactive, screen-oriented programs want this), after calling `initscr()` you should call `"cbreak(); noecho();"` Most programs would additionally call `"nonl(); intrflush (stdscr, FALSE); keypad(stdscr, TRUE);"`.

Before a *curses* program is run, a terminal's tab stops should be set and its initialization strings, if defined, must be output. To do this, execute `tput init` command after the shell environment variable `TERM` has been exported. For further details, see *profile(M)*, *tput(C)*, and the "Tabs and Initialization" subsection of *terminfo(M)*.

The *curses* library contains routines that manipulate data structures called *windows* that can be thought of as two-dimensional arrays of characters representing all or part of a terminal screen. A default window called `stdscr` is supplied, which is the size of the terminal screen. Others may be created with `newwin()`. Windows are referred to by variables declared as `WINDOW *`; the type `WINDOW` is defined in `<curses.h>` to be a structure. These data structures are manipulated with routines described below, among which the most basic are `move()` and `addch()`. (More general versions of these routines are included with names beginning with `w`, allowing you to specify a window. The routines not beginning with `w` usually affect `stdscr`.) Then



**refresh()** is called, telling the routines to make the user's terminal screen look like **stdscr**. The characters in a window are actually of type **chtype**, so that other information about the character may also be stored with each character.

Special windows called *pads* may also be manipulated. These are windows which are not constrained to the size of the screen and whose contents need not be displayed completely. See the description of **newpad()** under "Window and Pad Manipulation" for more information.

In addition to drawing characters on the screen, video attributes may be included which cause the characters to be underlined or shown in reverse video on terminals that support such display enhancements. Line drawing characters may be specified to be output. On input, *curses* is also able to translate arrow and function keys that transmit escape sequences into single values. The video attributes, line drawing characters, and input values use names, defined in `<curses.h>`, such as **A\_REVERSE**, **ACS\_HLINE**, and **KEY\_LEFT**.

Routines that manipulate color on color alphanumeric terminals are new in this release of *curses*. To use these routines **start\_color()** must be called, usually right after **initscr()**. Colors are always used in pairs (referred to as color-pairs). A color-pair consists of a foreground color (for characters) and a background color (for the field the characters are displayed on). A programmer initializes a color-pair with the routine **init\_pair()**. After it has been initialized, **COLOR\_PAIR(n)**, a macro defined in `<curses.h>`, can be used in the same ways other video attributes can be used. If a terminal is capable of redefining colors the programmer can use the routine **init\_color()** to change the definition of a color. The routines **has\_color()** and **can\_change\_color()** return **TRUE** or **FALSE**, depending on whether the terminal has color capabilities and whether the user can change the colors. The routine **color\_content()** allows a user to identify the amounts of red, green, and blue components in an initialized color. The routine **pair\_content()** allows a user to find out how a given color-pair is currently defined.

*curses* also defines the **WINDOW \*** variable, **curscr**, which is used only for certain low-level operations like clearing and redrawing a garbaged screen. **curscr** can be used in only a few routines. If the window argument to **clearok()** is **curscr**, the next call to **wrefresh()** with any window will cause the screen to be cleared and repainted from scratch. If the window argument to **wrefresh()** is **curscr**, the screen is immediately cleared and repainted from scratch. This is how most programs would implement a "repaint-screen" function. More information on using **curscr** is provided where its use is appropriate.

If the environment variable **TERMINFO** is defined, any program using *curses* will check for a local terminal definition before checking in the standard place. For example, if the environment variable **TERM** is set to **wy60**, then the compiled terminal definition is found in



`/usr/lib/terminfo/w/wy60`. (The `w` is copied from the first letter of `wy60` to avoid creation of huge directories.) However, if `TERMINFO` is set to `$HOME/myterms`, `curses` will first check `$HOME/myterms/w/wy60`, and, if that fails, will then check `/usr/lib/terminfo/w/wy60`. This is useful for developing experimental definitions or when write permission on `/usr/lib/terminfo` is not available.

The integer variables `LINES` and `COLS` are defined in `<curses.h>`, and will be filled in by `initscr()` with the size of the screen. (For more information, see the subsection "Terminfo-Level Manipulations.") The integer variables `COLORS` and `COLOR_PAIRS` are also defined in `<curses.h>` and contain, respectively, the maximum number of colors and color-pairs the terminal can support. They are initialized by `start_color()`. The constants `TRUE` and `FALSE` have the values `1` and `0`, respectively. The constants `ERR` and `OK` are returned by routines to indicate whether the routine successfully completed. These constants are also defined in `<curses.h>`.

## Routines

---

Many of the following routines have two or more versions. The routines prefixed with `w` require a *window* argument. The routines prefixed with `p` require a *pad* argument. Those without a prefix generally use `stdscr`.

The routines prefixed with `mv` require `y` and `x` coordinates to move to before performing the appropriate action. The `mv()` routines imply a call to `move()` before the call to the other routine. The window argument is always specified before the coordinates. `y` always refers to the row (of the window), and `x` always refers to the column. The upper left corner is always `(0,0)`, not `(1,1)`. The routines prefixed with `mvw` take both a *window* argument and `y` and `x` coordinates.

In each case, `win` is the window affected and `pad` is the pad affected. (`win` and `pad` are always of type `WINDOW *`.) Option-setting routines require a boolean flag `bf` with the value `TRUE` or `FALSE`. (`bf` is always of type `bool`.) The types `WINDOW`, `bool`, and `chtype` are defined in `<curses.h>`. See the Syntax for a summary of what types all variables are.

All routines return either the integer `ERR` or the integer `OK`, unless otherwise noted. Routines that return pointers always return `NULL` on error.

Sometimes the description of a routine refers to a second routine. If the routine referred to is prefixed with a `w`, then you should assume that other versions of the second routine behave similarly. For example, the description of `initscr()` refers to `wrefresh()`. This implies that the same result will occur if `refresh()` is called.

See the list above to determine which routines are termcap *curses* and which are terminfo *curses* only.

### Overall Screen Manipulation

- WINDOW \*initscr()** The first routine called should almost always be **initscr()**. (The exceptions are **slk\_init()**, **filter()**, and **ripoffline()**.) This will determine the terminal type and initialize all *curses* data structures. **initscr()** also arranges that the first call to **wrefresh()** will clear the screen. If errors occur, **initscr()** will write an appropriate error message to standard error and exit; otherwise, a pointer to **stdscr** is returned. If the program wants an indication of error conditions, **newterm()** should be used instead of **initscr()**. **initscr()** should only be called once per application.
- endwin()** A program should always call **endwin()** before exiting or escaping from *curses* mode temporarily, to do a shell escape or *system(S)* call, for example. This routine will restore *tty(M)* modes, move the cursor to the lower left corner of the screen and reset the terminal into the proper non-visual mode. To resume after a temporary escape, call **wrefresh()** or **doupdate()**.
- isendwin()** Returns **TRUE** if **endwin()** has been called without any subsequent calls to **wrefresh()**.
- SCREEN \*newterm(type, outfd, infd)** A program that outputs to more than one terminal must use **newterm()** for each terminal instead of **initscr()**. A program that wants an indication of error conditions, so that it may continue to run in a line-oriented mode if the terminal cannot support a screen-oriented program, must also use this routine. **newterm()** should be called once for each terminal. It returns a variable of type **SCREEN\*** that should be saved as a reference to that terminal. The arguments are the *type* of the terminal to be used in place of the environment variable **TERM**; *outfd*, a *stdio(S)* file pointer for output to the terminal; and *infd*, another file pointer for input from the terminal. When it is done running, the program must also call **endwin()** for each terminal being used. If **newterm()** is called



more than once for the same terminal, the first terminal referred to must be the last one for which **endwin()** is called.

#### **SCREEN \*set\_term(new)**

This routine is used to switch between different terminals. The screen reference *new* becomes the new current terminal. A pointer to the screen of the previous terminal is returned by the routine. This is the only routine which manipulates **SCREEN** pointers; all other routines affect only the current terminal.

### Window and Pad Manipulation

#### **refresh()** **wrefresh(win)**

These routines (or **prefresh()**, **pnoutrefresh()**, **wnoutrefresh()**, or **doupdate()**) must be called to write output to the terminal, as most other routines merely manipulate data structures. **wrefresh()** copies the named window to the physical terminal screen, taking into account what is already there in order to minimize the amount of information that's sent to the terminal (called optimization). **refresh()** does the same thing, except it uses **stdscr** as a default window. Unless **leaveok()** has been enabled, the physical cursor of the terminal is left at the location of the window's cursor. The number of characters output to the terminal is returned.

Note that **refresh()** is a macro.

#### **wnoutrefresh(win)** **doupdate()**

These two routines allow multiple updates to the physical terminal screen with more efficiency than **wrefresh()** alone. How this is accomplished is described in the next paragraph.

*curses* keeps two data structures representing the terminal screen: a *physical* terminal screen, describing what is actually on the screen, and a *virtual* terminal screen, describing what the programmer wants to have on the screen. **wrefresh()** works by first calling **wnoutrefresh()**, which copies the named window to the virtual screen, and then by calling **doupdate()**, which compares the virtual screen to the physical screen and does the actual update. If the programmer wishes to



output several windows at once, a series of calls to **wrefresh()** will result in alternating calls to **wnoutrefresh()** and **doupdate()**, causing several bursts of output to the screen. By first calling **wnoutrefresh()** for each window, it is then possible to call **doupdate()** once, resulting in only one burst of output, with probably fewer total characters transmitted and certainly less processor time used.

**WINDOW \*newwin**(*nlines*, *ncols*, *begin\_y*, *begin\_x*)

Create and return a pointer to a new window with the given number of lines (or rows), *nlines*, and columns, *ncols*. The upper left corner of the window is at line *begin\_y*, column *begin\_x*. If either *nlines* or *ncols* is 0, they will be set to the value of *lines-begin\_y* and *cols-begin\_x*. A new full-screen window is created by calling **newwin(0,0,0,0)**.

**mvwin**(*win*, *y*, *x*)

Move the window so that the upper left corner will be at position (*y*, *x*). If the move would cause any portion of the window to be moved off the screen, it is an error and the window is not moved.

**WINDOW \*subwin**(*orig*, *nlines*, *ncols*, *begin\_y*, *begin\_x*)

Create and return a pointer to a new window with the given number of lines (or rows), *nlines*, and columns, *ncols*. The window is at position (*begin\_y*, *begin\_x*) on the screen. (This position is relative to the screen, and not to the window *orig*.) The window is made in the middle of the window *orig*, so that changes made to one window will affect the character image of both windows. When changing the image of a subwindow, it will be necessary to call **touchwin()** or **touchline()** on *orig* before calling **wrefresh()** on *orig*.

**delwin**(*win*)

Delete the named window, freeing up all memory associated with it. If you try to delete a main window before all of its subwindows have been deleted, ERR will be returned.

**WINDOW \*newpad**(*nlines*, *ncols*)

Create and return a pointer to a new pad data structure with the given number of lines (or rows), *nlines*, and columns, *ncols*. A pad is a window that is not restricted by the screen

size and is not necessarily associated with a particular part of the screen. Pads can be used when a large window is needed, and only a part of the window will be on the screen at one time. Automatic refreshes of pads (for example, from scrolling or echoing of input) do not occur. It is not legal to call **wrefresh()** with a pad as an argument; the routines **prefresh()** or **pnoutrefresh()** should be called instead. Note that these routines require additional parameters to specify the part of the pad to be displayed and the location on the screen to be used for display.

**WINDOW \*subpad**(*orig*, *nlines*, *ncols*, *begin\_y*, *begin\_x*)

Create and return a pointer to a subwindow within a pad with the given number of lines (or rows), *nlines*, and columns, *ncols*. Unlike **subwin()**, which uses screen coordinates, the window is at position (*begin\_y*, *begin\_x*) on the pad. The window is made in the middle of the window *orig*, so that changes made to one window will affect the character image of both windows. When changing the image of a subwindow, it will be necessary to call **touchwin()** or **touchline()** on *orig* before calling **prefresh()** on *orig*.

**prefresh**(*pad*, *pminrow*, *pmincol*, *sminrow*, *smincol*, *smaxrow*, *smaxcol*)

**pnoutrefresh**(*pad*, *pminrow*, *pmincol*, *sminrow*, *smincol*, *smaxrow*, *smaxcol*)

These routines are analogous to **wrefresh()** and **wnoutrefresh()** except that pads, instead of windows, are involved. The additional parameters are needed to indicate what part of the pad and screen are involved. *pminrow* and *pmincol* specify the upper left corner, in the pad, of the rectangle to be displayed. *sminrow*, *smincol*, *smaxrow*, and *smaxcol* specify the edges, on the screen, of the rectangle to be displayed in. The lower right corner in the pad of the rectangle to be displayed is calculated from the screen coordinates, since the rectangles must be the same size. Both rectangles must be entirely contained within their respective structures. Negative values of *pminrow*, *pmincol*, *sminrow*, or *smincol* are treated as if they were zero.



**Output**

These routines are used to manipulate text in windows.

**addch**(ch)

**waddch**(win, ch)

**mvaddch**(y, x, ch)

**mvwaddch**(win, y, x, ch)

The character *ch* is put into the window at the current cursor position of the window and the position of the window cursor is advanced. Its function is similar to that of *putchar* (see *putc*(S)). At the right margin, an automatic newline is performed. At the bottom of the scrolling region, if **scrollok**() is enabled, the scrolling region will be scrolled up one line.

If *ch* is a tab, newline, or backspace, the cursor will be moved appropriately within the window. A newline also does a **wclrtoeol**() before moving. Tabs are considered to be at every eighth column. If *ch* is another control character, it will be drawn in the ^X notation. (Calling **winch**() on a position in the window containing a control character will not return the control character, but instead will return one character of the representation of the control character.)

Video attributes can be combined with a character by or-ing them into the parameter. This will result in these attributes also being set. (The intent here is that text, including attributes, can be copied from one place to another using **winch**() and **waddch**()). See **wstandout**(), below.

Note that *ch* is actually of type **chtype**, not a character.

Note that **addch**(), **mvaddch**(), and **mvwaddch**() are macros.

**echochar**(ch)

**wechochar**(win, ch)

**pechochar**(pad, ch)

These routines are functionally equivalent to a call to **addch**(ch) followed by a call to **refresh**(), a call to **waddch**(win, ch) followed by a call to **wrefresh**(win), or a call to **waddch**(pad, ch) followed by a call to **prefresh**(pad). The knowledge that only a single character is being output is taken into consideration and, for non-control characters, a considerable performance gain can be seen by using these routines instead of their



equivalents. In the case of **pechochar()**, the last location of the pad on the screen is reused for the arguments to **prefresh()**.

Note that *ch* is actually of type **chtype**, not a character.

Note that **echochar()** is a macro.

**addstr(str)**  
**waddstr(win, str)**  
**mvwaddstr(win, y, x, str)**  
**mvaddstr(y, x, str)**

These routines write all the characters of the null-terminated character string *str* on the given window. This is equivalent to calling **waddch()** once for each character in the string.

Note that **addstr()**, **mvaddstr()**, and **mvwaddstr()** are macros.

**attroff(attrs)**  
**wattroff(win, attrs)**  
**attron(attrs)**  
**wattron(win, attrs)**  
**attrset(attrs)**  
**wattrset(win, attrs)**  
**standend()**  
**wstandend(win)**  
**standout()**  
**wstandout(win)**

These routines manipulate the current attributes of the named window. These attributes can be any combination of the constants **A\_STANDOUT**, **A\_REVERSE**, **A\_BOLD**, **A\_DIM**, **A\_BLINK**, **A\_UNDERLINE**, and **A\_ALTCHARSET**, as well as the macro **COLOR\_PAIR(n)**. These attributes are defined in `< curses.h >` and can be combined with the C logical OR (**|**) operator.

The current attributes of a window are applied to all characters that are written into the window with **waddch()**. Attributes are a property of the character, and move with the character through any scrolling and insert/delete line/character operations. To the extent possible on the particular terminal, they will be displayed as the graphic rendition of the characters put on the screen.

**wattrset(win, attrs)** sets the current attributes of the given window to *attrs*. **wattroff(win, attrs)** turns off the named attributes without turning on or off any other attributes.

**wattron**(win, attrs) turns on the named attributes without affecting any others. **wstandout**(win, attrs) is the same as **wattron**(win, A\_STANDOUT). **wstandend**(win, attrs) is the same as **wattrset**(win, 0), that is, it turns off all attributes.

Note that **wattroff**(), **wattron**(), **wattrset**(), **wstandend**(), and **wstandout**() return 1 at all times.

Note that *attrs* is actually of type **chtype**, not a character.

Note that **attroff**(), **attron**(), **attrset**(), **standend**(), and **standout**() are macros.

**beep**()  
**flash**()

These routines are used to signal the terminal user. **beep**() will sound the audible alarm on the terminal, if possible, and if not, will flash the screen (visible bell), if that is possible. **flash**() will flash the screen, and if that is not possible, will sound the audible signal. If neither signal is possible, nothing will happen. Nearly all terminals have an audible signal (bell or beep) but only some can flash the screen.

**box**(win, vertch, horch) A box is drawn around the edge of the window, *win*. *vertch* and *horch* are the characters the box is to be drawn with. If *vertch* and *horch* are 0, then appropriate default characters, ACS\_VLINE and ACS\_HLINE, will be used.

Note that *vertch* and *horch* are actually of type **chtype**, not characters.

**erase**()  
**werase**(win)

These routines copy blanks to every position in the window.

Note that **erase**() is a macro.

**clear**()  
**wclear**(win)

These routines are like **erase**() and **werase**(), but they also call **clearok**(), arranging that the screen will be cleared completely on the next call to **wrefresh**() for that window, and repainted from scratch.

Note that **clear**() is a macro.

**clrtoobot()**  
**wclrtoobot(win)**

All lines below the cursor in this window are erased. Also, the current line to the right of the cursor, inclusive, is erased.

Note that **clrtoobot()** is a macro.

**clrtoeol()**  
**wclrtoeol(win)**

The current line to the right of the cursor, inclusive, is erased.

Note that **clrtoeol()** is a macro.

**delay\_output(ms)**

Insert a *ms* millisecond pause in the output. It is not recommended that this routine be used extensively, because padding characters are used rather than a processor pause.

**delch()**  
**wdelch(win)**  
**mvdelch(y, x)**  
**mvwdelch(win, y, x)**

The character under the cursor in the window is deleted. All characters to the right on the same line are moved to the left one position and the last character on the line is filled with a blank. The cursor position does not change (after moving to *(y, x)*, if specified). (This does not imply use of the hardware "delete-character" feature.)

Note that **delch()**, **mvdelch()**, and **mvwdelch()** are macros.

**deleteln()**  
**wdeleteln(win)**

The line under the cursor in the window is deleted. All lines below the current line are moved up one line. The bottom line of the window is cleared. The cursor position does not change. (This does not imply use of the hardware "delete-line" feature.)

Note that **deleteln()** is a macro.

**getyx(win, y, x)**

The cursor position of the window is placed in the two integer variables *y* and *x*.

Note that **getyx()** is a macro, so no "&" is necessary before the variables *y* and *x*.

**getbegyx(win, y, x)**  
**getmaxyx(win, y, x)**

The current beginning coordinates (**getbegyx()**) or size (**getmaxyx()**) of the specified window are placed in the two integer variables *y* and *x*.



Note that **getbegyx()** and **getmaxyx()** are macros, so no "&" is necessary before the variables *y* and *x*.

**getorg**(win, y, x)  
**getdim**(win, y, x)

These are the *termcap* functions that get the beginning coordinates and dimensions of the specified window.

**insch**(ch)  
**winsch**(win, ch)  
**mvwinsch**(win, y, x, ch)  
**mvinsch**(y, x, ch)

The character *ch* is inserted before the character under the cursor. All characters to the right are moved one space to the right, losing the rightmost character of the line. The cursor position does not change (after moving to (*y*, *x*), if specified). (This does not imply use of the hardware "insert-character" feature.)

Note that *ch* is actually of type **chtype**, not a character.

Note that **insch()**, **mvinsch()**, and **mvwinsch()** are macros.

**insertln**( )  
**winsertln**(win)

A blank line is inserted above the current line and the bottom line is lost. (This does not imply use of the hardware "insert-line" feature.)

Note that **insertln()** is a macro.

**move**(y, x)  
**wmove**(win, y, x)

The cursor associated with the window is moved to line (row) *y*, column *x*. This does not move the physical cursor of the terminal until **wrefresh()** is called. The position specified is relative to the upper left corner of the window, which is (0, 0).

Note that **move()** is a macro.

**overlay**(srcwin, dstwin)  
**overwrite**(srcwin, dstwin)

These routines overlay text from *srcwin* on top of text from *dstwin* wherever the two windows overlap. The difference is that **overlay()** is non-destructive (blanks are not copied), while **overwrite()** is destructive.

**copywin**(srcwin, dstwin, sminrow, smincol, dminrow, dmincol,  
dmaxrow, dmaxcol, overlay)

This routine provides finer control over the **overlay()** and **overwrite()** routines. As in the **prefresh()** routine, a rectangle is

specified in the destination window, (*dminrow*, *dmincol*) and (*dmaxrow*, *dmaxcol*), and the upper-left-corner coordinates of the source window, (*sminrow*, *smincol*). If the argument *overlay* is true, then copying is non-destructive, as in **overlay()**.

**printw**(fmt [, arg ...])

**wprintw**(win, fmt [, arg ...])

**mvprintw**(y, x, fmt [, arg ...])

**mvwprintw**(win, y, x, fmt [, arg ...])

These routines are analogous to *printf*(S). The string which would be output by *printf*(S) is instead output using **waddstr()** on the given window.

**vwprintw**(win, fmt, varlist)

This routine corresponds to *vfprintf*(S). It performs a **wprintw()** using a variable argument list. The third argument is a *va list*, a pointer to a list of arguments, as defined in *<varargs.h>*. See the *vfprintf*(S) and *varargs*(S) manual pages for a detailed description on how to use variable argument lists.

**scroll**(win)

The window is scrolled up one line. This involves moving the lines in the window data structure.

**touchwin**(win)

**touchline**(win, start, count)

Throw away all optimization information about which parts of the window have been touched, by pretending that the entire window has been drawn on. This is sometimes necessary when using overlapping windows, since a change to one window will affect the other window, but the records of which lines have been changed in the other window will not reflect the change. **touchline()** only pretends that *count* lines have been changed, beginning with line *start*.

## Input

**getch()**

**wgetch**(win)

**mvgetch**(y, x)

**mvwgetch**(win, y, x)

A character is read from the terminal associated with the window. In **NODELAY** mode, if there is no input waiting, the value **ERR** is



returned. In DELAY mode, the program will hang until the system passes text through to the program. Depending on the setting of **cbreak()**, this will be after one character (CBREAK mode), or after the first newline (NOCBREAK mode). In HALF-DELAY mode, the program will hang until a character is typed or the specified timeout has been reached. Unless **noecho()** has been set, the character will also be echoed into the designated window.

When **wgetch()** is called, before getting a character, it will call **wrefresh()** if anything in the window has changed (for example, the cursor has moved or text changed).

When using **getch()**, **wgetch()**, **mvwgetch()**, or **mvwgetch()**, do not set both NOCBREAK mode [**nocbreak()**] and ECHO mode [**echo()**] at the same time. Depending on the state of the *tty*(M) driver when each character is typed, the program may produce undesirable results.

If **wgetch()** encounters a **^D**, it is returned (unlike *stdio* routines, which would return a null string and have a return code of -1).

If **keypad(win, TRUE)** has been called, and a function key is pressed, the token for that function key will be returned instead of the raw characters. (See **keypad()** under "Input Options Setting.") Possible function keys are defined in `<curses.h>` with integers beginning with **0401**, whose names begin with **KEY\_**. If a character is received that could be the beginning of a function key (such as escape), *curses* will set a timer. If the remainder of the sequence is not received within the designated time, the character will be passed through, otherwise the function key value will be returned. For this reason, on many terminals, there will be a delay after a user presses the escape key before the escape is returned to the program. (Use by a programmer of the escape key for a single character routine is discouraged. Also see **notimeout()** below.)

Note that **getch()**, **mvwgetch()**, and **mvwgetch()** are macros.



**getstr**(str)  
**wgetstr**(win, str)  
**mvgetstr**(y, x, str)  
**mvwgetstr**(win, y, x, str)

A series of calls to **wgetch**() is made, until a newline, carriage return, or enter key is received. The resulting value (except for this terminating character) is placed in the area pointed at by the character pointer *str*. The user's erase and kill characters are interpreted. See **wgetch**() for how it handles characters differently from *stdio* routines (especially **^D**).

Note that **getstr**(), **mvgetstr**(), and **mvwgetstr**() are macros.

**ungetch**(c)

Place *c* onto the input queue, to be returned by the next call to **wgetch**().

**flushinp**()

Throws away any typeahead that has been typed by the user and has not yet been read by the program. Note that **flushinp**() will not throw away any characters supplied by **ungetch**().

**inch**()  
**winch**(win)  
**mvinch**(y, x)  
**mvwinch**(win, y, x)

The character, of type **chtype**, at the current position in the named window is returned. If any attributes are set for that position, their values will be OR'ed into the value returned. The predefined constants **A\_CHARTEXT** and **A\_ATTRIBUTES**, defined in *< curses.h >*, can be used with the C logical AND (&) operator to extract the character or attributes alone.

Note that **inch**(), **winch**(), **mvinch**(), and **mvwinch**() are macros.

**scanw**(fmt [, arg...])  
**wscanw**(win, fmt [, arg...])  
**mvscanw**(y, x, fmt [, arg...])  
**mvwscanw**(win, y, x, fmt [, arg...])

These routines correspond to *scanf*(S), as do their arguments and return values. **wgetstr**() is called on the window, and the resulting line is used as input for the scan. The return value for these routines is the number of *arg* values that are converted by *fmt*. *arg* values that are not converted are lost. See **wgetstr**() for how it handles strings differently from the *stdio* routines (especially **^D**).

**vwscanw**(win, fmt, ap) This routine is similar to **vwprintw()** in that it performs a **wscanw()** using a variable argument list. The third argument is a *va list*, a pointer to a list of arguments, as defined in *<varargs.h>*. See the *vprintf(S)* and *varargs(S)* manual pages for a detailed description on how to use variable argument lists.

### Output Options Setting

These routines set options within *curses* that deal with output. All options are initially FALSE, unless otherwise stated. It is not necessary to turn these options off before calling **endwin()**.

**clearok**(win, bf)

If enabled (*bf* is TRUE), the next call to **wrefresh()** with this window will clear the screen completely and redraw the entire screen from scratch. This is useful when the contents of the screen are uncertain, or in some cases for a more pleasing visual effect.

**idlok**(win, bf)

If enabled (*bf* is TRUE), *curses* will consider using the hardware "insert/delete-line" feature of terminals so equipped. If disabled (*bf* is FALSE), *curses* will very seldom use this feature. (The "insert/delete-character" feature is always considered.) This option should be enabled only if your application needs "insert/delete-line", for example, for a screen editor. It is disabled by default because "insert/delete-line" tends to be visually annoying when used in applications where it isn't really needed. If "insert/delete-line" cannot be used, *curses* will redraw the changed portions of all lines. Not calling **idlok()** saves approximately 5000 bytes of memory.

**leaveok**(win, bf)

Normally, the hardware cursor is left at the location of the window cursor being refreshed. This option allows the cursor to be left wherever the update happens to leave it. It is useful for applications where the cursor is not used, since it reduces the need for cursor motions. If possible, the cursor is made invisible when this option is enabled.



**setscrreg**(top, bot)  
**wsetscrreg**(win, top, bot)

These routines allow the user to set a software scrolling region in a window. *top* and *bot* are the line numbers of the top and bottom margin of the scrolling region. (Line 0 is the top line of the window.) If this option and **scrollok**( ) are enabled, an attempt to move off the bottom margin line will cause all lines in the scrolling region to scroll up one line. (Note that this has nothing to do with use of a physical scrolling region capability in the terminal, like that in the DEC VT100. Only the text of the window is scrolled; if **idlok**( ) is enabled and the terminal has either a scrolling region or "insert/delete-line" capability, they will probably be used by the output routines.)

Note that **setscrreg**( ) is a macro.

**scrollok**(win, bf)

This option controls what happens when the cursor of a window is moved off the edge of the window or scrolling region, either from a newline on the bottom line, or typing the last character of the last line. If disabled (*bf* is **FALSE**), the cursor is left on the bottom line at the location where the offending character was entered. If enabled (*bf* is **TRUE**), **wrefresh**( ) is called on the window, and then the physical terminal and window are scrolled up one line. (Note that in order to get the physical scrolling effect on the terminal, it is also necessary to call **idlok**( ).)

Note that **scrollok**( ) will always return **OK**.

### Input Options Setting

These routines set options within *curses* that deal with input. The options involve using *ioctl*(S) and therefore interact with *curses* routines. It is not necessary to turn these options off before calling **endwin**( ).

**cbreak**( )  
**crmode**( )  
**nocbreak**( )  
**nocrmode**( )

These routines put the terminal into and out of CBREAK mode, respectively. In CBREAK mode, characters typed by the user are immediately available to the program and erase/kill character processing is not



performed. When in NOCBREAK mode, the *tty* driver will buffer characters typed until a newline or carriage return is typed. Interrupt and flow-control characters are unaffected by this mode (see *termio*(M)). Initially the terminal may or may not be in CBREAK mode, as it is inherited, therefore, a program should explicitly call **cbreak()** or **crmode()** to be placed in CBREAK mode and **nocbreak()** or **nocrmode()** to be taken out of CBREAK mode. Most interactive programs using *curses* will set CBREAK mode.

Note that **cbreak()** performs a subset of the functionality of **raw()**. See **wgetch()** under "Input" for a discussion of how these routines interact with **echo()** and **noecho()**.

**echo()**  
**noecho()**

These routines control whether characters typed by the user are echoed by **wgetch()** as they are typed. Echoing by the *tty* driver is always disabled, but initially **wgetch()** is in ECHO mode, so characters typed are echoed. Authors of most interactive programs prefer to do their own echoing in a controlled area of the screen, or not to echo at all, so they disable echoing by calling **noecho()**. See **wgetch()** under "Input" for a discussion of how these routines interact with **cbreak()** and **nocbreak()**.

**nl()**  
**nonl()**

These routines control whether carriage return is translated into newline on input by **wgetch()**. Initially, this translation is done; **nonl()** turns the translation off. Note that translation by the *tty*(M) driver is disabled in CBREAK mode.

**halfdelay**(tenths)

Half-delay mode is similar to CBREAK mode in that characters typed by the user are immediately available to the program. However, after blocking for *tenths* tenths of seconds, **ERR** will be returned if nothing has been typed. *tenths* must be a number between 1 and 255. Use **nocbreak()** to leave half-delay mode.

**intrflush**(win, bf)

If this option is enabled, when an interrupt key is pressed on the keyboard (interrupt, break, quit) all output in the *tty* driver queue will be flushed, giving the effect of faster

response to the interrupt, but causing *curses* to have the wrong idea of what is on the screen. Disabling the option prevents the flush. The default for the option is inherited from the tty driver settings. The window argument is ignored.

**keypad**(win, bf)

This option enables *curses* to obtain information from the keypad of the user's terminal. If enabled, the user can press a function key (such as an arrow key) and **wgetch()** will return a single value representing the function key, as in **KEY\_LEFT**. If disabled, *curses* will not treat function keys specially and the program would have to interpret the escape sequences itself. If the keypad in the terminal can be turned on (made to transmit), calling **keypad** (win, TRUE) will turn it on.

**addkey**(str, val)

This routine is available in *termcap* curses only. To add additional sequences to the key-mode tree. *str* is the character sequence sent by the key. *val* is the integer to be returned when the key sequence is recognized on input.

**meta**(win, bf)

Initially, whether the terminal returns 7 or 8 significant bits on input depends on the control mode of the tty driver (see *termio*(M)). To force 8 bits to be returned, invoke **meta** (win, TRUE). To force 7 bits to be returned, invoke **meta** (win, FALSE). The window argument, *win*, is always ignored. If the *terminfo*(F) capabilities **smm** (meta\_on) and **rmm** (meta\_off) are defined for the terminal, **smm** will be sent to the terminal when **meta** (win, TRUE) is called and **rmm** will be sent when **meta** (win, FALSE) is called.

**nodelay**(win, bf)

This option causes **wgetch()** to be a non-blocking call. If no input is ready, **wgetch()** will return **ERR**. If disabled, **wgetch()** will hang until a key is pressed.

**notimeout**(win, bf)

While interpreting an input escape sequence, **wgetch()** will set a timer while waiting for the next character. If **notimeout**(win, TRUE) is called, then **wgetch()** will not set a timer. The purpose of the timeout is to differentiate between sequences received from a function key and those typed by a user.



**raw()**  
**noraw()**

The terminal is placed into or out of RAW mode. RAW mode is similar to CBREAK mode, in that characters typed are immediately passed through to the user program; however, in RAW mode, the interrupt, quit, suspend, and flow control characters are passed through uninterpreted, instead of generating a signal as they do in CBREAK mode. The behavior of the BREAK key depends on other bits in the *tty(M)* driver that are not set by *curses*.

**typeahead(fildes)**

*curses* does "line-breakout optimization" by looking for typeahead periodically while updating the screen. If input is found, and it is coming from a *tty*, the current update will be postponed until **wrefresh()** or **doupdate()** is called again. This allows faster response to commands typed in advance. Normally, the file descriptor for the input FILE pointer passed to **newterm()**, or **stdin** in the case that **initscr()** was used, will be used to do this typeahead checking. The **typeahead()** routine specifies that the file descriptor *fildes* is to be used to check for typeahead instead. If *fildes* is **-1**, then no typeahead checking will be done.

Note that *fildes* is a file descriptor, not a `<stdio.h>` FILE pointer.

### Environment Queries

**baudrate()**

Returns the output speed of the terminal. The number returned is in bits per second, for example, 9600, and is an integer.

**char erasechar()**

The user's current erase character is returned.

**has\_ic()**

True if the terminal has insert- and delete-character capabilities.

**has\_il()**

True if the terminal has insert- and delete-line capabilities, or can simulate them using scrolling regions. This might be used to check to see if it would be appropriate to turn on physical scrolling using **scrollok()** or **idlok()**.



- char killchar()** The user's current line-kill character is returned.
- char \*longname()** This routine returns a pointer to a static area containing a verbose description of the current terminal. The maximum length of a verbose description is 128 characters. It is defined only after the call to **initscr()** or **newterm()**. The area is overwritten by each call to **newterm()** and is not restored by **set\_term()**, so the value should be saved between calls to **newterm()** if **longname()** is going to be used with multiple terminals.

### Color Manipulation

This section describes the color manipulation routines introduced in this release of *curses*.

- can\_change\_color()** This routine requires no arguments. It returns **TRUE** if the terminal supports colors and can change their definitions, **FALSE** otherwise. This routine facilitates writing terminal-independent programs.

- color\_content(color, &r, &g, &b)** This routine gives users a way to find the intensity of the red, green, and blue (RGB) components in a color. It requires four arguments: the color number, and three addresses of **shorts** for storing the information about the amounts of red, green, and blue components in the given color. The value of the first argument must be between 0 and **COLORS-1**. The values that will be stored at the addresses pointed to by the last three arguments will be between 0 (no component) and 1000 (maximum amount of component). This routine returns **ERR** if the color does not exist (the first argument is outside the valid range), or if the terminal cannot change color definitions, **OK** otherwise.

- has\_colors()** This routine requires no arguments. It returns **TRUE** if the terminal can manipulate colors, **FALSE** otherwise. This routine facilitates writing terminal-independent programs. For example, a programmer can use it to decide whether to use color or some other video attribute.

**init\_color**(color, r, g, b) This routine changes the definition of a color. It takes four arguments: the number of the color to be changed followed by three RGB values (for the amounts of red, green, and blue components). (See the section **COLOR** for the default color index.) The value of the first argument must be between 0 and **COLORS-1**. The last three arguments must each be a value between 0 and 1000. When **init\_color()** is used, all occurrences of that color on the screen immediately change to the new definition. It returns **OK** if it was able to change the definition of the color, **ERR** otherwise.

**init\_pair**(pair, f, b) This routine changes the definition of a color-pair. It takes three arguments: the number of the color-pair to be changed, the foreground color number, and the background color number. The value of the first argument must be between 1 and **COLOR\_PAIRS-1**. The value of the second and third arguments must be between 0 and **COLORS-1**. If the color-pair was previously initialized, the screen will be refreshed and all occurrences of that color-pair will be changed to the new definition. The routine returns **OK** if it was able to change the definition of the color-pair, **ERR** otherwise.

**pair\_content**(pair, &f, &b) This routine allows users to find out what colors a given color-pair consists of. It requires three arguments: the color-pair number, and two addresses of **shorts** for storing the foreground and the background color numbers. The value of the first argument must be between 1 and **COLOR\_PAIRS-1**. The values that will be stored at the addresses pointed to by the second and third arguments will be between 0 and **COLORS-1**. The routine returns **ERR** if the color-pair has not been initialized, **OK** otherwise.

**start\_color()** This routine requires no arguments. It must be called if the user wants to use colors, and before any other color manipulation routine is called. It is good practice to call this routine right after **initscr()**. **start\_color()** initializes eight basic colors (black, blue, green, cyan, red, magenta, yellow, and white), and two global variables, **COLORS** and



**COLOR\_PAIRS** (respectively defining the maximum number of colors and color-pairs the terminal can support). It also restores the terminal's colors to the values they had when the terminal was just turned on. It returns **ERR** if the terminal does not support colors, **OK** otherwise.

### Soft Labels

If desired, *curses* will manipulate the set of soft function-key labels that exist on many terminals. For those terminals that do not have soft labels, if you want to simulate them, *curses* will take over the bottom line of **stdscr**, reducing the size of **stdscr** and the variable **LINES**. *curses* standardizes on 8 labels of 8 characters each. If a *curses* program changes the values of the soft labels, it can restore them only to the default settings for that terminal. Therefore, if before calling a *curses* program a user changes the values of the soft labels, those values cannot be reset when the *curses* program terminates.

#### **slk\_init(labfmt)**

In order to use soft labels, this routine must be called before **initscr()** or **newterm()** is called. If **initscr()** winds up using a line from **stdscr** to emulate the soft labels, then *labfmt* determines how the labels are arranged on the screen. Setting *labfmt* to **0** indicates that the labels are to be arranged in a 3-2-3 arrangement; **1** asks for a 4-4 arrangement.

#### **slk\_set(labnum, label, labfmt)**

*labnum* is the label number, from 1 to 8. *label* is the string to be put on the label, up to 8 characters in length. A **NULL** string or a **NULL** pointer will put up a blank label. *labfmt* is one of **0**, **1** or **2**, to indicate whether the label is to be left-justified, centered, or right-justified within the label.

#### **slk\_refresh()** **slk\_noutrefresh()**

These routines correspond to the routines **wrefresh()** and **wnoutrefresh()**. Most applications would use **slk\_noutrefresh()** because a **wrefresh()** will most likely soon follow.

#### **char \*slk\_label(labnum)**

The current label for label number *labnum* is returned, in the same format as it was in when it was passed to **slk\_set()**; that is, how it looked prior to being justified according to the *labfmt* argument of **slk\_set()**.



|                      |                                                                                                       |
|----------------------|-------------------------------------------------------------------------------------------------------|
| <b>slk_clear()</b>   | The soft labels are cleared from the screen.                                                          |
| <b>slk_restore()</b> | The soft labels are restored to the screen after a <b>slk_clear()</b> .                               |
| <b>slk_touch()</b>   | All of the soft labels are forced to be output the next time a <b>slk_noutrefresh()</b> is performed. |

### Low-Level *curses* Access

The following routines give low-level access to various *curses* functionality. These routines typically would be used inside of library routines.

|                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>def_prog_mode()</b><br><b>def_shell_mode()</b>     | Save the current terminal modes as the “program” (in <i>curses</i> ) or “shell” (not in <i>curses</i> ) state for use by the <b>reset_prog_mode()</b> and <b>reset_shell_mode()</b> routines. This is done automatically by <b>initscr()</b> .                                                                                                                                                                                                                                                                     |
| <b>reset_prog_mode()</b><br><b>reset_shell_mode()</b> | Restore the terminal to “program” (in <i>curses</i> ) or “shell” (out of <i>curses</i> ) state. These are done automatically by <b>endwin()</b> and <b>doupdate()</b> after an <b>endwin()</b> , so they normally would not be called.                                                                                                                                                                                                                                                                             |
| <b>set_tty()</b><br><b>reset_tty()</b>                | These routines are available only in <i>termcap</i> <i>curses</i> and perform the same functions as <b>reset_shell_mode()</b> and <b>reset_prog_mode()</b> .                                                                                                                                                                                                                                                                                                                                                       |
| <b>resetty()</b><br><b>savetty()</b>                  | These routines save and restore the state of the terminal modes. <b>savetty()</b> saves the current state of the terminal in a buffer and <b>resetty()</b> restores the state to what it was at the last call to <b>savetty()</b> .                                                                                                                                                                                                                                                                                |
| <b>getsyx(y, x)</b>                                   | The current coordinates of the virtual screen cursor are returned in <i>y</i> and <i>x</i> . If <b>leaveok()</b> is currently <b>TRUE</b> , then <b>-1,-1</b> will be returned. If lines have been removed from the top of the screen using <b>ripline()</b> , <i>y</i> and <i>x</i> include these lines; therefore, <i>y</i> and <i>x</i> should be used only as arguments for <b>setsyx()</b> .<br><br>Note that <b>getsyx()</b> is a macro, so no “&” is necessary before the variables <i>y</i> and <i>x</i> . |

**setsyx(y, x)**

The virtual screen cursor is set to *y*, *x*. If *y* and *x* are both **-1**, then **leaveok()** will be set. The two routines **getsyx()** and **setsyx()** are designed to be used by a library routine which manipulates *curses* windows but does not want to change the current position of the program's cursor. The library routine would call **getsyx()** at the beginning, do its manipulation of its own windows, do a **wnoutrefresh()** on its windows, call **setsyx()**, and then call **doupdate()**.

**ripoffline(line, init)**

This routine provides access to the same facility that **slk\_init()** uses to reduce the size of the screen. **ripoffline()** must be called before **initscr()** or **newterm()** is called. If *line* is positive, a line will be removed from the top of **stdscr**; if negative, a line will be removed from the bottom. When this is done inside **initscr()**, the routine **init()** is called with two arguments: a window pointer to the 1-line window that has been allocated and an integer with the number of columns in the window. Inside this initialization routine, the integer variables **LINES** and **COLS** (defined in *< curses.h >*) are not guaranteed to be accurate and **wrefresh()** or **doupdate()** must not be called. It is allowable to call **wnoutrefresh()** during the initialization routine.

**ripoffline()** can be called up to five times before calling **initscr()** or **newterm()**.

**scr\_dump(filename)**

The current contents of the virtual screen are written to the file *filename*.

**dmp\_win(win,fp,margin)**

This routine is available only through *termcap* *curses* and performs the same function as **scr\_dump()**.

**scr\_restore(filename)**

The virtual screen is set to the contents of *filename*, which must have been written using **scr\_dump()**. **ERR** is returned if the contents of *filename* are not compatible with the current release of *curses* software. The next call to **doupdate()** will restore the screen to what it looked like in the dump file.



**scr\_init**(filename)

The contents of *filename* are read in and used to initialize the *curses* data structures about what the terminal currently has on its screen. If the data is determined to be valid, *curses* will base its next update of the screen on this information rather than clearing the screen and starting from scratch. **scr\_init**() would be used after **initscr**() or a *system*(S) call to share the screen with another process which has done a **scr\_dump**() after its **endwin**() call. The data will be declared invalid if the *terminfo*(F) capability **nrrmc** is true or the time-stamp of the tty is old. Note that **keypad**(), **meta**(), **slk\_clear**(), **curs\_set**(), **flash**(), and **beep**() do not affect the contents of the screen, but will make the tty's time-stamp old.

**curs\_set**(visibility)

The cursor state is set to invisible, normal, or very visible for *visibility* equal to 0, 1 or 2. If the terminal supports the *visibility* requested, the previous *cursor* state is returned; otherwise, **ERR** is returned.

**curon**()**curoff**()**is\_curoff**()

These routines are available only through *termcap* curses and turn the cursor display on and off, respectively.

**draino**(ms)

Wait until the output has drained enough that it will only take *ms* more milliseconds to drain completely.

**garbagedlines**(win, begline, numlines)

This routine indicates to *curses* that a screen line is garbaged and should be thrown away before having anything written over the top of it. It could be used for programs such as editors which want a command to redraw just a single line. Such a command could be used in cases where there is a noisy communications line and redrawing the entire screen would be subject to even more communication noise. Just redrawing the single line gives some semblance of hope that it would show up unblemished. The current location of the window is used to determine which lines are to be redrawn.



**napms**(ms) Sleep for *ms* milliseconds.

**mvcur**(oldrow, oldcol, newrow, newcol)  
Low-level cursor motion.

### Terminfo-Level Manipulations

These low-level routines must be called by programs that need to deal directly with the *terminfo*(F) database to handle certain terminal capabilities, such as programming function keys. For all other functionality, *curses* routines are more suitable and their use is recommended.

Initially, **setupterm**() should be called. (Note that **setupterm**() is automatically called by **initscr**() and **newterm**()). This will define the set of terminal-dependent variables defined in the *terminfo*(F) database. The *terminfo*(F) variables **lines** and **columns** (see *terminfo*(F)) are initialized by **setupterm**() as follows: if the environment variables **LINES** and **COLUMNS** exist, their values are used. If the above environment variables do not exist and the program is running in a layer (see *layers*(C)), the size of the current layer is used. Otherwise, the values for **lines** and **columns** specified in the *terminfo*(F) database are used.

The header files **<curses.h>** and **<term.h>** should be included, in this order, to get the definitions for these strings, numbers, and flags. Parameterized strings should be passed through **tparm**() to instantiate them. All *terminfo*(F) strings (including the output of **tparm**()) should be printed with **tputs**() or **putp**(). Before exiting, **reset\_shell\_mode**() should be called to restore the tty modes. Programs which use cursor addressing should output **enter\_ca\_mode** upon startup and should output **exit\_ca\_mode** before exiting (see *terminfo*(F)). (Programs desiring shell escapes should call **reset\_shell\_mode**() and output **exit\_ca\_mode** before the shell is called and should output **enter\_ca\_mode** and call **reset\_prog\_mode**() after returning from the shell. Note that this is different from the *curses* routines (see **endwin**()).

**setupterm**(term, fildes, errret)

Reads in the *terminfo*(F) database, initializing the *terminfo*(F) structures, but does not set up the output virtualization structures used by *curses*. The terminal type is in the character string *term*; if *term* is **NULL**, the environment variable **TERM** will be used. All output is to the file descriptor *fildes*. If *errret* is not **NULL**, then **setupterm**() will return **OK** or **ERR** and store a status value in the integer pointed to by *errret*. A status of **1** in *errret* is normal, **0** means that the terminal could not be found, and **-1** means that the -

*terminfo*(F) database could not be found. If *errret* is **NULL**, **setupterm()** will print an error message upon finding an error and exit. Thus, the simplest call is **setupterm ((char \*)0, 1, (int \*)0)**, which uses all the defaults.

The *terminfo*(F) boolean, numeric and string variables are stored in a structure of type **TERMINAL**. After **setupterm()** returns successfully, the variable **cur\_term** (of type **TERMINAL \***) is initialized with all of the information that the *terminfo*(F) boolean, numeric and string variables refer to. The pointer may be saved before calling **setupterm()** again. Further calls to **setupterm()** will allocate new space rather than reuse the space pointed to by **cur\_term**.

**set\_curterm(nterm)**

*nterm* is of type **TERMINAL \***. **set\_curterm()** sets the variable **cur\_term** to *nterm*, and makes all of the *terminfo*(F) boolean, numeric and string variables use the values from *nterm*.

**del\_curterm(oterm)**

*oterm* is of type **TERMINAL \***. **del\_curterm()** frees the space pointed to by *oterm* and makes it available for further use. If *oterm* is the same as **cur\_term**, then references to any of the *terminfo*(F) boolean, numeric and string variables thereafter may refer to invalid memory locations until another **setupterm()** has been called.

**restartterm(term, fildes, errret)**

Similar to **setupterm()**, except that it is called after restoring memory to a previous state; for example, after a call to **scr\_restore()**. It assumes that the windows and the input and output options are the same as when memory was saved, but the terminal type and baud rate may be different.

**char \*tparm(str, p<sub>1</sub>, p<sub>2</sub>, ..., p<sub>9</sub>)**

Instantiate the string *str* with parms p<sub>1</sub>. A pointer is returned to the result of *str* with the parameters applied.

**tputs(str, count, putc)**

Apply padding to the string *str* and output it. *str* must be a *terminfo*(F) string variable or the return value from **tparm()**, **tgetstr()**, **tigetstr()** or **tgoto()**. *count* is the number of



lines affected, or **1** if not applicable. *putc* is a *putchar*(S)-like routine to which the characters are passed, one at a time.

- putp**(str)                      A routine that calls **tputs** (*str*, **1**, *putchar*).
- vidputs**(attrs, putc)        Output a string that puts the terminal in the video attribute mode *attrs*, which is any combination of the attributes listed below. The characters are passed to the *putchar*(S)-like routine *putc* ().
- vidattr**(attrs)              Similar to **vidputs** (), except that it outputs through *putchar*(S).

The following routines return the value of the capability corresponding to the character string containing the *terminfo*(F) *capname* passed to them. For example, **rc = tigetstr("acsc")** causes the value of *acsc* to be returned in *rc*.

- tigetflag**(capname)        The value **-1** is returned if *capname* is not a boolean capability. The value **0** is returned if *capname* is not defined for this terminal.
- tigetnum**(capname)        The value **-2** is returned if *capname* is not a numeric capability. The value **-1** is returned if *capname* is not defined for this terminal.
- tigetstr**(capname)        The value (char \*) **-1** is returned if *capname* is not a string capability. A null value is returned if *capname* is not defined for this terminal.

```
char *boolnames[], *boolcodes[], *boolfnames[]
char *numnames[], *numcodes[], *numfnames[]
char *strnames[], *strcodes[], *strfnames[]
```

These null-terminated arrays contain the *capnames*, the *termcap* codes, and the full C names, for each of the *terminfo*(F) variables.

### Termcap Emulation

These routines are included as a conversion aid for programs that use the *termcap* library. Their parameters are the same and the routines are emulated using the *terminfo*(F) database.

- tgetent**(bp, name)        Look up *termcap* entry for *name*. The emulation ignores the buffer pointer *bp*.



- tgetflag**(codename)    Get the boolean entry for *codename*.
- tgetnum**(codename)    Get numeric entry for *codename*.
- char \*tgetstr**(codename, area)  
     Return the string entry for *codename*. If *area* is not **NULL**, then also store it in the buffer pointed to by *area* and advance *area*. **tputs()** should be used to output the returned string.
- char \*tgoto**(cap, col, row)  
     Instantiate the parameters into the given capability. The output from this routine is to be passed to **tputs()**.
- tputs**(str, affcnt, putc)    See **tputs()** above, under “Terminfo-Level Manipulations.”

### Miscellaneous

- traceoff()**  
**traceon()**    Turn off and on debugging trace output when using the debug version of the *curses* library, */usr/lib/libdcurses.a*. This facility is available only to customers with a source license.
- unctrl**(c)    This macro expands to a character string which is a printable representation of the character *c*. Control characters are displayed in the ^X notation. Printing characters are displayed as is.
- unctrl()** is a macro, defined in *<unctrl.h>*, which is automatically included by *<curses.h>*.
- char \*keyname**(c)    A character string corresponding to the key *c* is returned.
- filter()**    This routine is one of the few that is to be called before **initscr()** or **newterm()** is called. It arranges things so that *curses* thinks that there is a 1-line screen. *curses* will not use any terminal capabilities that assume that they know what line on the screen the cursor is on.

### Use of **curscr**

The special window **curscr** can be used in only a few routines. If the window argument to **clearok()** is **curscr**, the next call to **wrefresh()** with any window will cause the screen to be cleared and repainted from scratch. If the window argument to **wrefresh()** is **curscr**, the screen is immediately cleared and repainted from scratch. (This is how most programs would implement a "repaint-screen" routine.) The source window argument to **overlay()**, **overwrite()**, and **copywin()** may be **curscr**, in which case the current contents of the virtual terminal screen will be accessed.

### Obsolete Calls

Various routines are provided to maintain compatibility in programs written for older versions of the curses library. These routines are all emulated as indicated below.

|                    |                                         |
|--------------------|-----------------------------------------|
| <b>crmode()</b>    | Replaced by <b>cbreak()</b> .           |
| <b>fixterm()</b>   | Replaced by <b>reset_prog_mode()</b> .  |
| <b>gettmode()</b>  | A no-op.                                |
| <b>nocrmode()</b>  | Replaced by <b>nocbreak()</b> .         |
| <b>resetterm()</b> | Replaced by <b>reset_shell_mode()</b> . |
| <b>saveterm()</b>  | Replaced by <b>def_prog_mode()</b> .    |
| <b>setterm()</b>   | Replaced by <b>setupterm()</b> .        |

### Attributes

---

The following video attributes, defined in `<curses.h>`, can be passed to the routines **wattron()**, **wattroff()**, and **wattrset()**, or OR'ed with the characters passed to **waddch()**.

|                     |                                   |
|---------------------|-----------------------------------|
| <b>A_STANDOUT</b>   | Terminal's best highlighting mode |
| <b>A_UNDERLINE</b>  | Underlining                       |
| <b>A_REVERSE</b>    | Reverse video                     |
| <b>A_BLINK</b>      | Blinking                          |
| <b>A_DIM</b>        | Half bright                       |
| <b>A_BOLD</b>       | Extra bright or bold              |
| <b>A_ALTCHARSET</b> | Alternate character set           |

|                    |                                                                                                      |
|--------------------|------------------------------------------------------------------------------------------------------|
| COLOR_PAIR(n)      | Color_pair defined in n (Note that this is a macro.)                                                 |
| A_CHARTEXT         | Bit-mask to extract character (described under <b>winch()</b> )                                      |
| A_ATTRIBUTES       | Bit-mask to extract attributes (described under <b>winch()</b> )                                     |
| A_NORMAL           | Bit mask to reset all attributes off<br>(for example: <b>wattrset</b> (win, A_NORMAL))               |
| A_COLOR            | Bit-mask to extract color_pair field information                                                     |
| PAIR_NUMBER(attrs) | Returns the pair number associated with the<br>COLOR_PAIR(n) attribute. (Note that this is a macro.) |

## Colors

---

In `< curses.h >` the following macros are defined to have the numeric value shown. These are the default colors. *curses* also assumes that color 0 (zero) is the default background color for all terminals.

|               |   |
|---------------|---|
| COLOR_BLACK   | 0 |
| COLOR_BLUE    | 1 |
| COLOR_GREEN   | 2 |
| COLOR_CYAN    | 3 |
| COLOR_RED     | 4 |
| COLOR_MAGENTA | 5 |
| COLOR_YELLOW  | 6 |
| COLOR_WHITE   | 7 |

## Function Keys

---

The following function keys, defined in `< curses.h >`, might be returned by **wgetch()** if **keypad()** has been enabled. Note that not all of these may be supported on a particular terminal if the terminal does not transmit a unique code when the key is pressed or the definition for the key is not present in the *terminfo* (F) database.

| Name          | Value | Key name                                         |
|---------------|-------|--------------------------------------------------|
| KEY_BREAK     | 0401  | break key (unreliable)                           |
| KEY_DOWN      | 0402  | The four arrow keys ...                          |
| KEY_UP        | 0403  |                                                  |
| KEY_LEFT      | 0404  |                                                  |
| KEY_RIGHT     | 0405  | ...                                              |
| KEY_HOME      | 0406  | Home key (upward+left arrow)                     |
| KEY_BACKSPACE | 0407  | backspace (unreliable)                           |
| KEY_F0        | 0410  | Function keys. Space for 64 keys<br>is reserved. |



## CURSES (S)

## CURSES (S)

|              |              |                                                                   |
|--------------|--------------|-------------------------------------------------------------------|
| KEY_F(n)     | (KEY_F0+(n)) | Formula for f <sub>n</sub> .                                      |
| KEY_DL       | 0510         | Delete line                                                       |
| KEY_IL       | 0511         | Insert line                                                       |
| KEY_DC       | 0512         | Delete character                                                  |
| KEY_IC       | 0513         | Insert char or enter insert mode                                  |
| KEY_EIC      | 0514         | Exit insert char mode                                             |
| KEY_CLEAR    | 0515         | Clear screen                                                      |
| KEY_EOS      | 0516         | Clear to end of screen                                            |
| KEY_EOL      | 0517         | Clear to end of line                                              |
| KEY_SF       | 0520         | Scroll 1 line forward                                             |
| KEY_SR       | 0521         | Scroll 1 line backwards (reverse)                                 |
| KEY_NPAGE    | 0522         | Next page                                                         |
| KEY.PAGE     | 0523         | Previous page                                                     |
| KEY_STAB     | 0524         | Set tab                                                           |
| KEY_CTAB     | 0525         | Clear tab                                                         |
| KEY_CATAB    | 0526         | Clear all tabs                                                    |
| KEY_ENTER    | 0527         | Enter or send                                                     |
| KEY_SRESET   | 0530         | soft (partial) reset                                              |
| KEY_RESET    | 0531         | reset or hard reset                                               |
| KEY_PRINT    | 0532         | print or copy                                                     |
| KEY_LL       | 0533         | home down or bottom (lower left)<br>keypad is arranged like this: |
|              |              | A1 up A3                                                          |
|              |              | left B2 right                                                     |
|              |              | C1 down C3                                                        |
| KEY_A1       | 0534         | Upper left of keypad                                              |
| KEY_A3       | 0535         | Upper right of keypad                                             |
| KEY_B2       | 0536         | Center of keypad                                                  |
| KEY_C1       | 0537         | Lower left of keypad                                              |
| KEY_C3       | 0540         | Lower right of keypad                                             |
| KEY_BTAB     | 0541         | Back tab key                                                      |
| KEY_BEG      | 0542         | beg(inning) key                                                   |
| KEY_CANCEL   | 0543         | cancel key                                                        |
| KEY_CLOSE    | 0544         | close key                                                         |
| KEY_COMMAND  | 0545         | cmd (command) key                                                 |
| KEY_COPY     | 0546         | copy key                                                          |
| KEY_CREATE   | 0547         | create key                                                        |
| KEY_END      | 0550         | end key                                                           |
| KEY_EXIT     | 0551         | exit key                                                          |
| KEY_FIND     | 0552         | find key                                                          |
| KEY_HELP     | 0553         | help key                                                          |
| KEY_MARK     | 0554         | mark key                                                          |
| KEY_MESSAGE  | 0555         | message key                                                       |
| KEY_MOVE     | 0556         | move key                                                          |
| KEY_NEXT     | 0557         | next object key                                                   |
| KEY_OPEN     | 0560         | open key                                                          |
| KEY_OPTIONS  | 0561         | options key                                                       |
| KEY_PREVIOUS | 0562         | previous object key                                               |

|               |      |                         |
|---------------|------|-------------------------|
| KEY_REDO      | 0563 | redo key                |
| KEY_REFERENCE | 0564 | ref(erence) key         |
| KEY_REFRESH   | 0565 | refresh key             |
| KEY_REPLACE   | 0566 | replace key             |
| KEY_RESTART   | 0567 | restart key             |
| KEY_RESUME    | 0570 | resume key              |
| KEY_SAVE      | 0571 | save key                |
| KEY_SBEG      | 0572 | shifted beginning key   |
| KEY_SCANCEL   | 0573 | shifted cancel key      |
| KEY_SCOMMAND  | 0574 | shifted command key     |
| KEY_SCOPY     | 0575 | shifted copy key        |
| KEY_SCREATE   | 0576 | shifted create key      |
| KEY_SDC       | 0577 | shifted delete char key |
| KEY_SDL       | 0600 | shifted delete line key |
| KEY_SELECT    | 0601 | select key              |
| KEY_SEND      | 0602 | shifted end key         |
| KEY_SEOL      | 0603 | shifted clear line key  |
| KEY_SEXIT     | 0604 | shifted exit key        |
| KEY_SFIND     | 0605 | shifted find key        |
| KEY_SHELP     | 0606 | shifted help key        |
| KEY_SHOME     | 0607 | shifted home key        |
| KEY_SIC       | 0610 | shifted input key       |
| KEY_SLEFT     | 0611 | shifted left arrow key  |
| KEY_SMESSAGE  | 0612 | shifted message key     |
| KEY_SMOVE     | 0613 | shifted move key        |
| KEY_SNEXT     | 0614 | shifted next key        |
| KEY_SOPTIONS  | 0615 | shifted options key     |
| KEY_SPREVIOUS | 0616 | shifted prev key        |
| KEY_SPRINT    | 0617 | shifted print key       |
| KEY_SREDO     | 0620 | shifted redo key        |
| KEY_SREPLACE  | 0621 | shifted replace key     |
| KEY_SRIGHT    | 0622 | shifted right arrow     |
| KEY_SRSUME    | 0623 | shifted resume key      |
| KEY_SSAVE     | 0624 | shifted save key        |
| KEY_SSUSPEND  | 0625 | shifted suspend key     |
| KEY_SUNDO     | 0626 | shifted undo key        |
| KEY_SUSPEND   | 0627 | suspend key             |
| KEY_UNDO      | 0630 | undo key                |

## Line Graphics

---

The following variables may be used to add line-drawing characters to the screen with `waddch()`. When defined for the terminal, the variable will have the `A_ALTCHARSET` bit turned on. Otherwise, the default character listed below will be stored in the variable. The names were chosen to be consistent with the DEC VT100 nomenclature.



| <i>Name</i>  | <i>Default</i> | <i>Glyph Description</i> |
|--------------|----------------|--------------------------|
| ACS_ULCORNER | +              | upper left corner        |
| ACS_LLCORNER | +              | lower left corner        |
| ACS_URCORNER | +              | upper right corner       |
| ACS_LRCORNER | +              | lower right corner       |
| ACS_RTEE     | +              | right tee (┘)            |
| ACS_LTEE     | +              | left tee (└)             |
| ACS_BTEE     | +              | bottom tee (┴)           |
| ACS_TTEE     | +              | top tee (┬)              |
| ACS_HLINE    | -              | horizontal line          |
| ACS_VLINE    |                | vertical line            |
| ACS_PLUS     | +              | plus                     |
| ACS_S1       | -              | scan line 1              |
| ACS_S9       | -              | scan line 9              |
| ACS_DIAMOND  | +              | diamond                  |
| ACS_CKBOARD  | :              | checker board (stipple)  |
| ACS_DEGREE   | '              | degree symbol            |
| ACS_PLMINUS  | #              | plus/minus               |
| ACS_BULLET   | o              | bullet                   |
| ACS_LARROW   | <              | arrow pointing left      |
| ACS_RARROW   | >              | arrow pointing right     |
| ACS_DARROW   | v              | arrow pointing down      |
| ACS_UARROW   | ^              | arrow pointing up        |
| ACS_BOARD    | #              | board of squares         |
| ACS_LANTERN  | #              | lantern symbol           |
| ACS_BLOCK    | #              | solid square block       |

## See Also

---

cc(CP), ld(CP), ioctl(S), plot(S), putc(S), scanf(S), stdio(S), system(S),  
vprintf(S), profile(M), term(F), terminfo(F), varargs(S).  
termio(M), tty(M).

## Diagnostics

---

All routines return the integer **OK** upon successful completion and the integer **ERR** upon failure, unless otherwise noted in the preceding routine descriptions.

All macros return the value of their **w** version, except **getsyx()**, **getyx()**, **getbegyx()**, **getmaxyx()**. For these macros, no useful value is returned.

Routines that return pointers always return (**type \***) **NULL** on error.



## Warnings

---

The plotting library *plot(S)* and the *curses* library *curses(S)* both use the names **erase()** and **move()**. The *curses* versions are macros. If you need both libraries, put the *plot(S)* code in a different source file from the *curses(S)* code, and/or **#undef move()** and **erase()** in the *plot(S)* code.

Between the time a call to **initscr()** and **endwin()** has been issued, use only the routines in the *curses* library to generate output. Using system calls or the “standard I/O package” (see *stdio(S)*) for output during that time can cause unpredictable results.

If a pointer passed to a routine as a window argument is null or out of range, the results are undefined (core may be dumped).

## Notes

---

Currently typeahead checking is done using a nodelay read followed by an **ungetch()** of any character that may have been read. Typeahead checking is done only if **wgetch()** has been called at least once. This may change when proper kernel support is available. Programs which use a mixture of their own input routines with *curses* input routines may wish to call **typeahead(-1)** to turn off typeahead checking.

The argument to **napms()** is currently rounded up to the nearest second.

**draino** (ms) only works for *ms* equal to 0.

## Standards Conformance

---

*curses* is conformant with:

AT&T SVID Issue 2, Select Code 307-127.

## cuserid

---

get character login name of the user

### Syntax

---

```
#include <stdio.h>
```

```
char *cuserid (s)
char *s;
```

### Description

---

The *cuserid* function generates a character-string representation of the login name that the owner of the current process is logged in under. If *s* is a NULL pointer, this representation is generated in an internal static area, the address of which is returned. Otherwise, *s* is assumed to point to an array of at least **L\_cuserid** characters; the representation is left in this array. The constant **L\_cuserid** is defined in the `<stdio.h>` header file.

### See Also

---

`getlogin(S)`, `getpwent(S)`.

### Diagnostics

---

If the login name cannot be found, *cuserid* returns a NULL pointer; if *s* is not a NULL pointer, a null character ('\0') will be placed at *s*[0].

### Standards Conformance

---

*cuserid* is conformant with:

The X/Open Portability Guide II of January 1987;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;  
and NIST FIPS 151-1.

## **dblock**

---

lock the entire Authentication database

### **Syntax**

---

```
#include <sys/types.h>
#include <sys/security.h>
#include <sys/audit.h>
#include <prot.h>
```

```
void ensure_db_lock (decibel)
int decibel;
```

```
int db_lock (decibel)
int decibel;
```

```
int db_unlock (decibel)
int decibel;
```

```
int file_lock (attempts, delay, fd, decibel)
register int attempts;
int delay;
int fd;
int decibel;
```

```
int file_unlock (name, fd, decibel)
register char *name;
register int fd;
int decibel;
```

```
void enter_quiet_zone ()
```

```
void exit_quiet_zone ()
```

```
int make_transition_files (pathname, ptemppathname, poldpath-
name)
register char *pathname;
register char **ptemppathname;
register char **poldpathname;
```

```
int replace_file (temppathname, pathname, oldpathname)
register char *temppathname;
register char *pathname;
register char *oldpathname;
```

### **Description**

---

These routines support update locking to the authentication database.



*Ensure db lock* guarantees that the database is locked now. The process will exit if the lock cannot occur. If the *decibel* argument is **AUTH\_QUIET**, the locking will be done without interruptions from signals. *Db lock* tries to lock the database. If the *decibel* argument is **AUTH\_QUIET**, the locking will be done without interruptions from signals. It returns 1 if the lock occurred and 0 if the lock could not be done. *Db unlock* removes a lock previously placed by *db\_lock*. If the *decibel* argument is **AUTH\_QUIET**, the signal states saved from *ensure db lock* or *db\_lock* are restored. (The *decibel* argument must match when creating and removing the same lock.) *db unlock* returns 1 if it successfully worked and 0 if there were no lock or the lock could not be removed.

The more generic *file lock* routine will lock the file opened as *fd*. It will try for *attempts* times, spaced at *delay* seconds apart. If the *decibel* argument is **AUTH\_QUIET**, the locking will be done without interruptions from signals. If the file can be locked, the routine returns a 1; otherwise, it returns a 0. The companion routine *file unlock* unlocks the file with *name* and descriptor *fd* and returns 1 if successful and 0 if not. The *decibel* argument used as in the *db\_unlock* case.

The *enter quiet zone* and *exit quiet zone* routines provide a way to enter and exit regions of code where the process is undisturbed by all external signals. This is helpful on sensitive regions of code where the security state is being altered and to disrupt the entire operation would leave the system in an insecure state or a state from which it is hard to recover.

*Enter quiet zone* turns off all keyboard signals, namely **SIGHUP**, **SIGINT** and **SIGQUIT**. It saves the previous state of each of those signals.

*Exit quiet zone* must be called after a call to *enter quiet zone*. *Exit quiet zone* restores those signal states set by *enter quiet zone*.

The only legal way to use these routines is in the sequence:

```
enter_quiet_zone();
...
exit_quiet_zone();
...
enter_quiet_zone();
...
exit_quiet_zone();
...
enter_quiet_zone();
...
exit_quiet_zone();
...
```

The routines *make\_transition\_files* and *replace\_file* are used together to update one of the authentication database files. *Make\_transition\_files* takes the input argument *pathname* and creates two new file names. The file names are returned to the reference of pointers *poldpathname* and *ptemppathname*. It is guaranteed that these two names cannot appear as database entries due to characters they use in the name. *Poldpathname* refers to the name in which to place the existing file referenced by *pathname* should there be trouble. *Ptemppathname* refers to the new file being constructed to replace *pathname*. Note that *malloc(S)* is used to create space for *poldpathname* and *ptemppathname* and that files are not actually opened or otherwise referenced by *make\_transition\_files*. A return value of 1 means the names were created and a return of 0 means they were not.

*Replace\_file* uses the same arguments to manipulate the actual files. It expects that *pathname* references the existing file, *ptemppathname* references the newly created file, and *poldpathname* references the place *pathname* will be moved to should an error occur in placing the new file in *pathname*. A return value of 1 means the file movements worked successfully and 0 means they were not. *Replace\_file* removes the space allocated for the names in *make\_transition\_files*.

## Notes

---

These routines only work as advertised when *set\_auth\_parameters* is called as the first item in *main()*.

## Files

---

/tcdb/files/auth/lock

## See Also

---

lockf(S), signal(S), malloc(S)

## dbm: dbminit, fetch, store, delete, firstkey, nextkey

---

performs database functions

### Syntax

---

```
#include <dbm.h>

typedef struct { char *dptr; int dsize; } datum;

int dbminit(file)
char *file;

datum fetch(key)
datum key;

int store(key, content)
datum key, content;

int delete(key)
datum key;

datum firstkey();

datum nextkey(key);
datum key;
```

### Description

---

These functions maintain key/content pairs in a database. The functions will handle very large (a billion blocks) databases and will access a keyed item in one or two file system accesses. The functions are obtained with the loader option **-ldb**.

*keys* and *contents* are described by the *datum* typedef. A *datum* specifies a string of *dsize* bytes pointed to by *dptr*. Arbitrary binary data, as well as normal ASCII strings, are allowed. The database is stored in two files. One file is a directory containing a bit map and has **.dir** as its suffix. The second file contains all data and has **.pag** as its suffix.

Before a database can be accessed, it must be opened by *dbminit*. At the time of this call, the files *file.dir* and *file.pag* must exist. (An empty database is created by creating zero-length **.dir** and **.pag** files.)



Once open, the data stored under a key is accessed by *fetch* and data is placed under a key by *store*. A key (and its associated contents) is deleted by *delete*. A linear pass through all keys in a database may be made, in an (apparently) random order, by use of *firstkey* and *nextkey*. *firstkey* will return the first key in the database. With any key *nextkey* will return the next key in the database. This code will traverse the database:

```
for(key=firstkey(); key.dptr!=NULL; key=nextkey(key))
```

## Example

---

The example program below uses dbm's dbmopen, store, fetch and delete functions. It reads in keys and data from a datafile (shown below) and stores them in a database called "testfile". Newlines ("n") are used for delimiters. It then reads the keys from "datafile" and uses them to fetch the data out, print the data, and delete the record.

To run this program the files "testfile.dir" and "testfile.pag" must exist and be empty (0 bytes). The "datafile" and test program are as follows:

```
101
UNIX Programming
105
System Administration
234
Intro to UNIX

101
105
234

#include <stdio.h>
#include <dbm.h>
#define KEYSIZE 5
#define DATASIZE 25

typedef struct adatum
{
 char *ptr;
 int size;
}datum;

datum key, data, testdata;

FILE *fp, *fopen();

char keybuf[KEYSIZE];
char keybuf2[KEYSIZE];
char databuf1[DATASIZE];
```

```

main()
{
 datum fetch();
 datum store();
 char c;

 /* Initialize the database */
 dbmopen("testfile");
 fp = fopen("datafile", "r");
 /* Read in Keys and Data until a newline */
 while ((c = getc(fp)) != '\n') {
 /* Read in a key */
 key.ptr = keybuf;
 *key.ptr++ = c;
 key.size = 1;
 while ((c = getc(fp)) != '\n') {
 *key.ptr++ = c;
 key.size++;
 }

 /* Read in a data field */
 data.size = 0;
 data.ptr = databuf1;
 while ((c = getc(fp)) != '\n') {
 *data.ptr++ = c;
 data.size++;
 }
 *data.ptr = '\0';
 data.size++;

 /* Store a record in the testfile database */
 data.ptr = databuf1;
 key.ptr = keybuf;
 printf("datasize %d keysize %d\n", data.size, key.size);
 printf("dataptr %s keyptr %s\n", data.ptr, key.ptr);
 store(key, data);
 }
 key.ptr = keybuf2;

 /* Read in keys from datafile,
 and use them to fetch records */
 while ((*key.ptr++ = getc(fp)) != EOF) {
 key.size = 1;
 while ((c = getc(fp)) != '\n') {
 *key.ptr++ = c;
 key.size++;
 }
 key.ptr = keybuf2;

 /* Fetch record specified by key */
 testdata = fetch(key);
 printf("Key: %s Data: %s\n", key.ptr, testdata.ptr);
 /* Delete the record */
 delete(key);
 /* Attempt to retrieve deleted record */
 testdata = fetch(key);
 }
}

```

```
/* printf to show data is now null */
 printf("Deleted Key: %s Data: %s\n",key.ptr,testdata.ptr);
}
}
```

## Diagnostics

---

All functions that return an *int* indicate errors with negative values. A zero return indicates ok. Routines that return a *datum* indicate errors with a null (0) *dptr*.

## Notes

---

The *.pag* file will contain holes so that its apparent size is about four times its actual content. Older version of some systems may create real file blocks for these holes when touched. These files cannot be copied by normal means (*cp*, *cat*, *tp*, *tar*, *ar*) without filling in the holes.

*dptr* pointers returned by these subroutines point into static storage that is changed by subsequent calls.

The sum of the sizes of a key/content pair must not exceed the internal block size (currently 1024 bytes). Moreover all key/content pairs that hash together must fit on a single block. *store* will return an error in the event that a disk block fills with inseparable data.

*delete* does not physically reclaim file space, although it does make it available for reuse.

The order of keys presented by *firstkey* and *nextkey* depends on a hashing function.

These routines are not reentrant, so they should not be used on more than one database at a time.

## Credit

---

This utility was developed at the University of California at Berkeley and is used with permission.



## defopen, defread

---

reads default entries

### Syntax

---

```
int defopen(filename)
char *filename;
```

```
char *defread(pattern)
char *pattern;
```

### Description

---

*defopen* and *defread* are a pair of routines designed to allow easy access to default definition files. System V/386 is normally distributed in binary form; the use of default files allows OEMs or site administrators to customize utility defaults without having the source code.

*defopen* opens the default file named by the pathname in *filename*. *defopen* returns null if it is successful in opening the file, or the *fopen* failure code (*errno*) if the open fails.

*defread* reads the previously opened file from the beginning until it encounters a line beginning with *pattern*. *defread* then returns a pointer to the first character in the line after the initial *pattern*. If a trailing newline character is read it is replaced by a null byte.

When all items of interest have been extracted from the opened file the program may call *defopen* with the name of another file to be searched, or it may call *defopen* with NULL, which closes the default file without opening another.

### Files

---

The UNIX style convention is for a system program *xyz* to store its defaults (if any) in the file */etc/default/xyz*.

### Diagnostics

---

*defopen* returns zero on success and nonzero if the open fails. The return value is the *errno* value set by *fopen* (S).

*defread* returns NULL if a default file is not open, if the indicated pattern could not be found, or if it encounters any line in the file greater than the maximum length of 128 characters.

## Notes

---

The return value points to static data, whose contents are overwritten by each call.

## Value Added

---

*defopen* and *defread* are extensions of AT&T System V provided by the Santa Cruz Operation.

## dial

establish an outgoing terminal line connection

### Syntax

```
#include <dial.h>
```

```
int dial (call)
CALL call;
```

```
void undial (fd)
int fd;
```

### Description

*dial* returns a file-descriptor for a terminal line open for read/write. The argument to *dial* is a CALL structure (defined in the <dial.h> header file).

When finished with the terminal line, the calling program must invoke *undial* to release the semaphore that has been set during the allocation of the terminal device.

The definition of CALL in the <dial.h> header file is:

```
typedef struct {
 struct termio *attr; /* pointer to termio attribute struct */
 int baud; /* transmission data rate */
 int speed; /* 212A modem: low=300, high=1200 */
 char *line; /* device name for outgoing line */
 char *telno; /* pointer to tel-no digits string */
 int modem; /* specify modem control
 for direct lines */
 char *device; /* unused */
 int dev_len; /* unused */
} CALL;
```

The CALL element *speed* is intended only for use with an outgoing dialed call, in which case its value should be either 300 or 1200 to identify the 113A modem, or the high- or low-speed setting on the 212A modem. Note that the 113A modem or the low-speed setting of the 212A modem will transmit at any rate between 0 and 300 bits per second. However, the high-speed setting of the 212A modem transmits and receives at 1200 bits per second only. The CALL element *baud* is for the desired transmission baud rate. For example, one might set *baud* to 110 and *speed* to 300 (or 1200). However, if *speed* is set to 1200, *baud* must be set to high (1200).



If the desired terminal line is a direct line, a string pointer to its device-name should be placed in the *line* element in the CALL structure. Legal values for such terminal device names are kept in the *Devices* file. In this case, the value of the *baud* element should be set to -1. This will cause **dial** to determine the correct value from the *Devices* file.

The *telno* element is for a pointer to a character string representing the telephone number to be dialed. Such numbers may consist only of these characters:

- 0-9 dial 0-9
- \* dial \*
- # dial #
- = wait for secondary dial tone
- delay for approximately 4 seconds

The CALL element *modem* is used to specify modem control for direct lines. This element should be non-zero if modem control is required. The CALL element *attr* is a pointer to a *termio* structure, as defined in the `<termio.h>` header file. A NULL value for this pointer element may be passed to the *dial* function, but if such a structure is included, the elements specified in it will be set for the outgoing terminal line before the connection is established. This is often important for certain attributes such as parity and baud-rate.

The CALL elements *device* and *dev\_len* are no longer used. They are retained in the CALL structure for compatibility reasons.

## Files

---

```
/usr/lib/uucp/Devices
/usr/lib/uucp/Systems
/usr/spool/locks/LCK..tty-device
```

## See Also

---

alarm(S), read(S), write(S).  
uucp(C), termio(M) in the *User's Reference*.

## Diagnostics

---

On failure, a negative value indicating the reason for the failure will be returned. Mnemonics for the negative indices as listed here are defined in the `<dial.h>` header file.

|         |     |                                             |
|---------|-----|---------------------------------------------|
| INTRPT  | -1  | /* interrupt occurred */                    |
| D_HUNG  | -2  | /* dialer hung (no return from write) */    |
| NO_ANS  | -3  | /* no answer within 10 seconds */           |
| ILL_BD  | -4  | /* illegal baud-rate */                     |
| A_PROB  | -5  | /* acu problem (open() failure) */          |
| L_PROB  | -6  | /* line problem (open() failure) */         |
| NO_Ldv  | -7  | /* can't open <i>Devices</i> file */        |
| DV_NT_A | -8  | /* requested device not available */        |
| DV_NT_K | -9  | /* requested device not known */            |
| NO_BD_A | -10 | /* no device available at requested baud */ |
| NO_BD_K | -11 | /* no device known at requested baud */     |
| DV_NT_E | -12 | /* requested speed does not match */        |
| BAD_SYS | -13 | /* system not in <i>Systems</i> file */     |

## Warnings

---

Including the `<dial.h>` header file automatically includes the `<termio.h>` header file.

The above routine uses `<stdio.h>`. This causes the routine to increase the size of programs that are not using standard I/O more than might be expected.

## Notes

---

An *alarm*(S) system call for 3600 seconds is made (and caught) within the *dial* module for the purpose of "touching" the *LCK..* file and constitutes the device allocation semaphore for the terminal device. Otherwise, *uucp*(C) may simply delete the *LCK..* entry on its 90-minute clean-up rounds. The alarm may go off while the user program is in a *read*(S) or *write*(S) system call, causing an apparent error return. If the user program expects to be around for an hour or more, error returns from *reads* should be checked for (`errno==EINTR`), and the *read* possibly reissued.

## difftime

---

computes the difference between time values

### Syntax

---

```
#include <time.h>
```

```
double difftime(time2, time1)
```

```
time_t time2;
```

```
time_t time1;
```

### Description

---

The *difftime* function computes the difference of *time2* - *time1*.

### Return Value

---

The *difftime* function returns the elapsed time in seconds from *time1* to *time2* as a double-precision number.

### See Also

---

time(S)

### Example

---

```
#include <time.h>
```

```
int mark[10000];
```

```
main()
```

```
{
```

```
 time_t start, finish;
```

```
 register int i, loop, n, num, step;
```

```
 printf("This program will take about 3 minutes ");
```

```
 printf("on an AT and 8 on a PC.\n Working...\n");
```

```
 time(&start);
```



```
for (loop = 0; loop < 1000; ++loop)
 for (num = 0, n = 3; n < 10000; n += 2)
 if (!mark[n])
 {
 /* printf("%d\n",n); */
 step = 2*n;
 for (i = 3*n; i < 10000; i += step)
 mark[i] = -1;
 ++num;
 }
time(&finish);

/* Prints average of 1000 loops through "sieve": */
printf("\nProgram takes %f seconds to find %d primes.\n",
diffime(finish, start), num);
}
```

### Output:

Program takes 0.482000 seconds to find 1228 primes.

This program calculates the amount of time needed to find the prime numbers between 3 and 10,000. To display the prime numbers, delete the outermost loop and the comment delimiters around the expression `printf("%d",n);`.

## Standards Conformance

---

*diffime* is conformant with:

ANSI X3.159-198X C Language Draft Standard, May 13, 1988.

# **directory: opendir, readdir, telldir, seekdir, rewinddir, closedir**

directory operations

## **Syntax**

```
#include <sys/types.h>
#include <dirent.h>
```

```
DIR *opendir (filename)
char *filename;
```

```
struct dirent *readdir (dirp)
DIR *dirp;
```

```
long telldir (dirp)
DIR *dirp;
```

```
void seekdir (dirp, loc)
DIR *dirp;
long loc;
```

```
void rewinddir (dirp)
DIR *dirp;
```

```
void closedir(dirp)
DIR *dirp;
```

## **Description**

*opendir* opens the directory named by *filename* and associates a *directory stream* with it. *opendir* returns a pointer to be used to identify the *directory stream* in subsequent operations. The pointer NULL is returned if *filename* cannot be accessed or is not a directory, or if it cannot *malloc* enough memory to hold a DIR structure or a buffer for the directory entries.

*readdir* returns a pointer to the next active directory entry. No inactive entries are returned. It returns NULL upon reaching the end of the directory or upon detecting an invalid location in the directory.

*telldir* returns the current location associated with the named *directory stream*.

*seekdir* sets the position of the next *readdir* operation on the *directory stream*. The new position, *loc*, is the position associated with the *directory stream* when the *telldir* operation was performed. Values

returned by *telldir* are good only if the directory has not changed due to compaction or expansion. This is not a problem with System V, but it may be with some file system types.

*rewinddir* resets the position of the named *directory stream* to the beginning of the directory.

*closedir* closes the named *directory stream* and frees the DIR structure.

The following errors can occur as a result of these operations.

*opendir*:

- [ENOTDIR]      A component of *filename* is not a directory.
- [EACCES]      A component of *filename* denies search permission.
- [EMFILE]      The maximum number of file descriptors are currently open.
- [EFAULT]      *filename* points outside the allocated address space.

*readdir*:

- [ENOENT]      The current file pointer for the directory is not located at a valid entry.
- [EBADF]      The file descriptor determined by the DIR stream is no longer valid. This results if the DIR stream has been closed.

*telldir*, *seekdir*, and *closedir*:

- [EBADF]      The file descriptor determined by the DIR stream is no longer valid. This results if the DIR stream has been closed.

## Example

---

Sample code which searches a directory for entry *name*:



```

dirp = opendir(".");
while ((dp = readdir(dirp)) != NULL)
 if (strcmp(dp->d_name, name) == 0)
 {
 closedir(dirp);
 return FOUND;
 }
closedir(dirp);
return NOT_FOUND;

```

## See Also

---

getdents(S), dirent(F)

## Warnings

---

*rewinddir* is implemented as a macro, so its function address cannot be taken.

## Standards Conformance

---

*closedir*, *opendir*, *readdir* and *rewinddir* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
 The X/Open Portability Guide II of January 1987;  
 IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
 System Support;  
 and NIST FIPS 151-1.

*seekdir* and *telldir* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
 and The X/Open Portability Guide II of January 1987.

## discr

---

check discretionary attributes of files and programs

### Syntax

---

```
#include <sys/types.h>
#include <sys/security.h>
#include <sys/audit.h>
#include <prot.h>

int privileged_user ();

void setuid_least_privilege (cmd)
char *cmd;

void ensure_e_ids (cmd)
char *cmd;

void ensure_discr_file (cmd, path)
char *cmd;
char *path;

int get_discr_file (path, attrs)
char *path;
struct discr_attr *attrs;

int discr_check_file (path)
char *path;
```

### Description

---

These routines provide a means to check the discretionary attributes of programs and files against the values kept in the authentication database. They make use of the “discr\_attr” structure in <prot.h> defined as:

```
struct discr_attr {
 unsigned short da_owner;
 unsigned short da_group;
 unsigned short da_mode;
 short da_wc_owner;
 short da_wc_group;
};
```

The *da\_owner*, *da\_group*, and *da\_mode* are the same as their counterparts found in *stat*(S). The *da\_wc\_owner* (*da\_wc\_group*) flag is 1 when the database specifies a wild-card (\*) for the owner (group) of the file. In this case, the *da\_owner* (*da\_group*) is to be ignored and

*any* owner (group) will satisfy the requirements of the database. When the flag is 0, the wild-card is not in effect and the *da\_owner* (*da\_group*) value is to be believed.

*privileged\_user* returns 0 if the user does not have special privileges associated with the program and returns 1 if the user has special privileges. This boolean routine can be used to determine how much information needs to be presented to the user or what types of options the user has to the program.

*setuid\_least\_privilege* is used to reset effective UID and/or GID values to those of their real counterparts. It is used after all the activities required of the *setuid* effective permissions (e.g., open files, create files, signal users) has been accomplished. By closing down the extra privilege(s) granted by the effective IDs, the principle of "least privilege" is aided if not met totally.

*ensure\_discr\_file* checks the *path* as it exists now in the file system against the discretionary values (owner, group, mode) in the authentication database. If the requirements of the database are not met, the *cmd* program is first audited and then exited.

*get\_discr\_file* returns the discretionary attributes of a file as stored in the authentication database. These values may not be the same as those stored with the actual file.

*discr\_check\_file* performs discretionary checking on the file designated by *path* and returns its results. If the result is 0, the file matches the settings in the authentication database exactly. If the result is positive, the file does not meet the requirements of the authentication database. If the result is negative, the file exceeds the minimum requirements of the authentication database (e.g., the mode on disk is more restrictive than that specified in the database).

## Notes

---

These routines only work as advertised when *set\_auth\_parameters* from *identity* (S) is called as the first item in *main*().

## Files

---

/tcb/files/auth/\*/\*

## See Also

---

stat(S), authcap(S), identity(S), authaudit(S)



**Value Added**

---

*discr* is an extension of AT&T System V provided by the Santa Cruz Operation.

## div

divides integers

---

### Syntax

---

```
#include <stdlib.h>

struct div_t {
 int quot; /* Quotient */
 int rem; /* Remainder */
} div(numerator, denominator)

int numerator;
int denominator;
```

### Description

---

The *div* function divides *numerator* by *denominator*, computing the quotient and the remainder. The sign of the quotient is the same as that of the mathematical quotient. Its absolute value is the largest integer that is less than the absolute value of the mathematical quotient. If the denominator is zero, the program terminates with an error message.

### Return Value

---

The *div* function returns a structure of type *div\_t*, comprising both the quotient and the remainder. The structure is defined in *stdlib.h*.

### See Also

---

ldiv(S)

## Example

---

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

main(argc, argv)
int argc;
char **argv;
{
 int x,y;
 div_t div_result;
 x = atoi(argv[1]);
 y = atoi(argv[2]);
 printf("x is %d, y is %d\n", x,y);
 div_result = div(x,y);
 printf("The quotient is %d, and the remainder is %d\n",
 div_result.quot, div_result.rem);
}
```

The example above takes two integers as command line arguments and displays the results of the integer division. This program accepts two arguments on the command line following the program name, then calls *div* to divide the first argument by the second. Finally, it prints the structure members *quot* and *rem*.

Assuming the executable file is named *tdiv*, it might be executed as:

**tdiv 5 2**

the output would read:

x is 5, y is 2  
The quotient is 2, and the remainder is 1

## Standards Conformance

---

*div* is conformant with:

ANSI X3.159-198X C Language Draft Standard, May 13, 1988.



# **drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48**

---

generate uniformly distributed pseudo-random numbers

## **Syntax**

---

**double drand48 ( )**

**double erand48 (xsubi)  
unsigned short xsubi[3];**

**long lrand48 ( )**

**long nrand48 (xsubi)  
unsigned short xsubi[3];**

**long mrand48 ( )**

**long jrand48 (xsubi)  
unsigned short xsubi[3];**

**void srand48 (seedval)  
long seedval;**

**unsigned short \*seed48 (seed16v)  
unsigned short seed16v[3];**

**void lcong48 (param)  
unsigned short param[7];**

## **Description**

---

This family of functions generates pseudo-random numbers using the well-known linear congruential algorithm and 48-bit integer arithmetic.

Functions *drand48* and *erand48* return non-negative double-precision floating-point values uniformly distributed over the interval  $[0.0, 1.0)$ .

Functions *lrand48* and *nrand48* return non-negative long integers uniformly distributed over the interval  $[0, 2^{31})$ .

Functions *mrnd48* and *jrnd48* return signed long integers uniformly distributed over the interval  $[-2^{31}, 2^{31})$ .

Functions *srnd48*, *seed48*, and *lcong48* are initialization entry points, one of which should be invoked before either *drand48*, *lrnd48*, or *mrnd48* is called. (Although it is not recommended practice, constant default initializer values will be supplied automatically if *drand48*, *lrnd48*, or *mrnd48* is called without a prior call to an initialization entry point.) Functions *erand48*, *nrnd48*, and *jrnd48* do not require an initialization entry point to be called first.

All the routines work by generating a sequence of 48-bit integer values,  $X_i$ , according to the linear congruential formula

$$X_{n+1} = (aX_n + c)_{\text{mod } m} \quad n \geq 0.$$

The parameter  $m = 2^{48}$ ; hence 48-bit integer arithmetic is performed. Unless *lcong48* has been invoked, the multiplier value  $a$  and the addend value  $c$  are given by

$$\begin{aligned} a &= 5\text{DEECE66D}_{16} = 273673163155_8 \\ c &= \text{B}_{16} = 13_8. \end{aligned}$$

The value returned by any of the functions *drand48*, *erand48*, *lrnd48*, *nrnd48*, *mrnd48*, or *jrnd48* is computed by first generating the next 48-bit  $X_i$  in the sequence. Then the appropriate number of bits, according to the type of data item to be returned, are copied from the high-order (leftmost) bits of  $X_i$  and transformed into the returned value.

The functions *drand48*, *lrnd48*, and *mrnd48* store the last 48-bit  $X_i$  generated in an internal buffer, and must be initialized prior to being invoked. The functions *erand48*, *nrnd48*, and *jrnd48* require the calling program to provide storage for the successive  $X_i$  values in the array specified as an argument when the functions are invoked. These routines do not have to be initialized; the calling program must place the desired initial value of  $X_i$  into the array and pass it as an argument. By using different arguments, functions *erand48*, *nrnd48*, and *jrnd48* allow separate modules of a large program to generate several independent streams of pseudo-random numbers; that is, the sequence of numbers in each stream will *not* depend upon how many times the routines have been called to generate numbers for the other streams.

The initializer function *srnd48* sets the high-order 32 bits of  $X_i$  to the 32 bits contained in its argument. The low-order 16 bits of  $X_i$  are set to the arbitrary value  $330\text{E}_{16}$ .

The initializer function *seed48* sets the value of  $X_i$  to the 48-bit value specified in the argument array. In addition, the previous value of  $X_i$  is copied into a 48-bit internal buffer used only by *seed48*, and a pointer to this buffer is the value returned by *seed48*. This returned pointer, which can just be ignored if not needed, is useful if a program



is to be restarted from a given point at some future time — use the pointer to get at and store the last  $X_i$  value, and then use this value to reinitialize via *seed48* when the program is restarted.

The initialization function *lcong48* allows the user to specify the initial  $X_i$ , the multiplier value  $a$ , and the addend value  $c$ . Argument array elements *param*[0-2] specify  $X_i$ , *param*[3-5] specify the multiplier  $a$ , and *param*[6] specifies the 16-bit addend  $c$ . After *lcong48* has been called, a subsequent call to either *srand48* or *seed48* will restore the “standard” multiplier and addend values,  $a$  and  $c$ , specified on the previous page.

## See Also

---

*rand*(S)

## Notes

---

The source code for the portable version can be used on computers which do not have floating-point arithmetic. In such a situation, functions *drand48* and *erand48* are replaced by the two new functions below.

**long *irand48* (m)  
unsigned short m;**

**long *krand48* (xsubi, m)  
unsigned short xsubi[3], m;**

Functions *irand48* and *krand48* return non-negative long integers uniformly distributed over the interval  $[0, m-1]$ .

## Standards Conformance

---

*drand48*, *erand48*, *jrand48*, *lrand48*, *mrand48*, *nrand48*, *seed48* and *srand48* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

*lcong48* is conformant with:

AT&T SVID Issue 2, Select Code 307-127.



## dup

duplicate an open file descriptor

### Syntax

```
int dup (fildes)
int fildes;
```

### Description

The *fildes* argument is a file descriptor obtained from a *creat*, *open*, *dup*, *fcntl*, or *pipe* system call. The *dup* system call returns a new file descriptor having the following in common with the original:

Same open file (or pipe)

Same file pointer (that is, both file descriptors share one file pointer)

Same access mode (read, write, or read/write)

The new file descriptor is set to remain open across *exec* system calls (see *fcntl*(S)).

The file descriptor returned is the lowest one available.

The *dup* system call will fail if one or more of the following is true:

- |           |                                                                                        |
|-----------|----------------------------------------------------------------------------------------|
| [EBADF]   | The <i>fildes</i> argument is not a valid open file descriptor.                        |
| [EINTR]   | A signal was caught during the <i>dup</i> system call.                                 |
| [EMFILE]  | NOFiles file descriptors are currently open.                                           |
| [ENOLINK] | <i>fildes</i> is on a remote machine and the link to that machine is no longer active. |

### See Also

*close*(S), *creat*(S), *exec*(S), *fcntl*(S), *open*(S), *pipe*(S), *lockf*(S)

## Diagnostics

---

Upon successful completion a non-negative integer, namely the file descriptor, is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## Standards Conformance

---

*dup* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

## dup2

---

duplicate an open file descriptor

### Syntax

---

```
int dup2 (fildes, fildes2)
int fildes, fildes2;
```

### Description

---

The *fildes* argument is a file descriptor referring to an open file, and *fildes2* is a non-negative integer less than NOFiles. *dup2* causes *fildes2* to refer to the same file as *fildes*. If *fildes2* already referred to an open file, it is closed first.

The *dup2* function will fail if one or more of the following is true:

- |          |                                                    |
|----------|----------------------------------------------------|
| [EBADF]  | <i>fildes</i> is not a valid open file descriptor. |
| [EMFILE] | NOFiles file descriptors are currently open.       |

### See Also

---

creat(S), close(S), exec(S), fcntl(S), open(S), pipe(S), lockf(S).

### Diagnostics

---

Upon successful completion a non-negative integer, namely the file descriptor, is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

### Standards Conformance

---

*dup2* is conformant with:

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.



## **ecvt, fcvt, gcvt**

convert floating-point number to string

---

### **Syntax**

---

```
char *ecvt (value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;
```

```
char *fcvt (value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;
```

```
char *gcvt (value, ndigit, buf)
double value;
int ndigit;
char *buf;
```

### **Description**

---

The *ecvt* function converts *value* to a null-terminated string of *ndigit* digits and returns a pointer thereto. The high-order digit is non-zero, unless the value is zero. The low-order digit is rounded. The position of the decimal point relative to the beginning of the string is stored indirectly through *decpt* (negative means to the left of the returned digits). The decimal point is not included in the returned string. If the sign of the result is negative, the word pointed to by *sign* is non-zero, otherwise it is zero.

*fcvt* is identical to *ecvt*, except that the correct digit has been rounded for printf “%f” (FORTRAN F-format) output of the number of digits specified by *ndigit*.

*gcvt* converts the *value* to a null-terminated string in the array pointed to by *buf* and returns *buf*. It attempts to produce *ndigit* significant digits in FORTRAN F-format if possible, otherwise E-format, ready for printing. A minus sign, if there is one, or a decimal point will be included as part of the returned string. Trailing zeros are suppressed.

### **See Also**

---

printf(S).

## Notes

---

The values returned by *ecvt* and *fcvt* point to a single static data array whose content is overwritten by each call.

## Standards Conformance

---

*ecvt*, *fcvt* and *gcvt* are conformant with:

The X/Open Portability Guide II of January 1987.

## **end, etext, edata**

---

last locations in program

### **Syntax**

---

```
extern end;
extern etext;
extern edata;
```

### **Description**

---

These names refer neither to routines nor to locations with interesting contents. The address of *etext* is the first address above the program text, *edata* above the initialized data region, and *end* above the uninitialized data region.

When execution begins, the program break (the first location beyond the data) coincides with *end*, but the program break may be reset by the routines of *brk(S)*, *malloc(S)*, standard input/output (*stdio(S)*), the profile (-p) option of *cc(CP)*, and so on. Thus, the current value of the program break should be determined by *sbrk ((char \*)0)* (see *brk(S)*).

### **See Also**

---

*cc(CP)*, *brk(S)*, *malloc(S)*, *stdio(S)*

### **Standards Conformance**

---

*edata*, *end*, and *etext* are conformant with:

The X/Open Portability Guide II of January 1987.



## erf, erfc

error function and complementary error function

### Syntax

```
#include <math.h>
```

```
double erf (x)
double x;
```

```
double erfc (x)
double x;
```

### Description

The *erf* function returns the error function of  $x$ , defined as 
$$\pm \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

*erfc*, which returns  $1.0 - \text{erf}(x)$ , is provided because of the extreme loss of relative accuracy if *erf*( $x$ ) is called for large  $x$  and the result subtracted from 1.0 (e.g., for  $x = 5$ , 12 places are lost).

### See Also

*exp*(S)

### Standards Conformance

*erf* and *erfc* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## ev\_block

---

wait until the queue contains an event

### Syntax

---

```
#include <types.h>
#include <param.h>
#include <sysmacros.h>
#include <page.h>
#include <sys/event.h>
#include <mouse.h>
```

```
int ev_block()
```

### Description

---

After a process has opened an event queue with *ev\_init(S)* and *ev\_open(S)*, *ev\_block* causes the process to sleep until there is an event in the event queue.

### Diagnostics

---

A call to *ev\_block* returns -1 if the process does not have an open event queue, or if it is interrupted. It returns zero if it succeeds.

### See Also

---

*ev\_close(S)*, *ev\_count(S)*, *ev\_flush(S)*, *ev\_getdev(S)*, *ev\_getemask(S)*, *ev\_gindev(S)*, *ev\_init(S)*, *ev\_open(S)*, *ev\_pop(S)*, *ev\_read(S)*, *ev\_resume(S)*, *ev\_setemask(S)*, *ev\_suspend(S)*

### Notes

---

This routine must be linked in with the **-levent** linker option.

## ev\_close

---

close the event queue and all associated devices

### Syntax

---

```
#include <types.h>
#include <param.h>
#include <sysmacros.h>
#include <page.h>
#include <sys/event.h>
#include <mouse.h>
```

```
int ev_close()
```

### Description

---

*ev\_close* closes the event queue and any event devices currently open. This call takes no arguments.

An event queue must have been opened previously with *ev\_init(S)* and *ev\_open(S)*.

### Diagnostics

---

This routine returns a negative number to indicate an error. Making this call before obtaining an open event queue is an example of such an error.

### See Also

---

*ev\_block(S)*, *ev\_count(S)*, *ev\_flush(S)*, *ev\_getdev(S)*, *ev\_getemask(S)*, *ev\_gindev(S)*, *ev\_init(S)*, *ev\_open(S)*, *ev\_pop(S)*, *ev\_read(S)*, *ev\_resume(S)*, *ev\_setemask(S)*, *ev\_suspend(S)*

### Notes

---

This routine must be linked in with the **-levent** linker option.



## ev\_count

---

returns the number of events currently in the queue

### Syntax

---

```
#include <types.h>
#include <param.h>
#include <sysmacros.h>
#include <page.h>
#include <sys/event.h>
#include <mouse.h>
```

```
int ev_count()
```

### Description

---

*ev\_count* Returns the number of events currently in the queue. An event queue must have been opened with *ev\_init(S)* and *ev\_open(S)*.

### Diagnostics

---

*ev\_count* returns -1 if there is not an open event queue.

### See Also

---

*ev\_block(S)*, *ev\_close(S)*, *ev\_flush(S)*, *ev\_getdev(S)*, *ev\_getemask(S)*, *ev\_gindev(S)*, *ev\_init(S)*, *ev\_open(S)*, *ev\_pop(S)*, *ev\_read(S)*, *ev\_resume(S)*, *ev\_setemask(S)*, *ev\_suspend(S)*

### Notes

---

This routine must be linked in with the **-levent** linker option.

## ev\_flush

---

discard all events currently in the queue

### Syntax

---

```
#include <types.h>
#include <param.h>
#include <sysmacros.h>
#include <page.h>
#include <sys/event.h>
#include <mouse.h>
```

```
int ev_flush()
```

### Description

---

*ev\_flush* discards all events currently in the queue. Events in the queue when *ev\_flush* is invoked will not be available to the program.

### Diagnostics

---

*ev\_flush* returns -1 if there is no open event queue. Normally it returns zero.

### See Also

---

*ev\_block(S)*, *ev\_close(S)*, *ev\_count(S)*, *ev\_getdev(S)*,  
*ev\_getemask(S)*, *ev\_gindev(S)*, *ev\_init(S)*, *ev\_open(S)*, *ev\_pop(S)*,  
*ev\_read(S)*, *ev\_resume(S)*, *ev\_setemask(S)*, *ev\_suspend(S)*

### Notes

---

This routine must be linked in with the **-levent** linker option.

## ev\_getdev

gets a list of devices feeding an event queue

### Syntax

```
#include <types.h>
#include <param.h>
#include <sysmacros.h>
#include <page.h>
#include <sys/event.h>
#include <mouse.h>

struct devinfo * getdev(dev_mask, devinfo)
dmask_t dev_mask;
struct devinfo *devinfo;
```

### Description

*ev\_getdev* allows a program to examine the devices that are attached to its event queue. An open event queue must have been previously obtained with *ev\_init(S)* and *ev\_open(S)*. This routine takes two arguments, a bitmask of device classes and a pointer to a device\_info structure. The device mask indicates the classes of devices in which the program is interested. The device pointer is used to cycle through the devices attached to the queue.

The device mask is made by OR'ing together a subset of **D\_REL**, **D\_ABS**, **D\_STRING** and **D\_OTHER**. These values represent classes of graphics input devices. **D\_REL** refers to relative locator devices like mice. **D\_ABS** refers to absolute locator devices like bitpads and lightpens. **D\_STRING** refers to character stream devices like the keyboard.

The device pointer parameter is NULL for the first call. Each call returns a pointer which should be passed in in subsequent calls. When the routine has iterated through all the devices attached to the queue, it returns NULL.

The device information pointer points to a structure which looks like this:



```

struct devinfo {
 short handle; /* not used by application */
 short class; /* REL, ABS, STRING or OTHER */
 short type; /* The type of hardware */
 char *name; /* Device name, from data files */
 char *key; /* Device key, from data files */
};

```

An application can examine this information and decide whether or not to use the device. The *ev\_gindev(S)* routine allows a program to exclude or later re-include a device. The pointer returned by *ev\_getdev* is passed in to *ev\_gindev(S)*.

When a queue is opened, a bitmask specifying what kinds of devices to attach is supplied. All devices of a class which is masked in are attached to the queue. This routine is used to examine those devices.

## Diagnostics

---

This routine returns -1 to a program which does not have an open event queue. It returns -2 if no devices of any class which is masked in are found. Normally it returns zero.

## See Also

---

*ev\_block(S)*, *ev\_close(S)*, *ev\_count(S)*, *ev\_flush(S)*, *ev\_getemask(S)*, *ev\_gindev(S)*, *ev\_init(S)*, *ev\_open(S)*, *ev\_pop(S)*, *ev\_read(S)*, *ev\_resume(S)*, *ev\_setemask(S)*, *ev\_suspend(S)*

## Notes

---

This routine must be linked in with the **-levent** linker option.

The keyboard is attached to an event queue whenever devices of class **D\_STRING** are requested. If the keyboard is attached to an event queue, then the keyboard will not generate normal stdin input, until the event queue is closed.

## ev\_gindev

---

include/exclude devices for event input

### Syntax

---

```
#include <types.h>
#include <param.h>
#include <sysmacros.h>
#include <page.h>
#include <sys/event.h>
#include <mouse.h>

int ev_gindev(devinfo, action)
struct devinfo *devinfo;
char action;
```

### Description

---

*ev\_gindev* is used in concert with *ev\_getdev* to exclude or later re-include devices from feeding the event queue. The argument *devinfo* contains the pointer to the device to be included or excluded. The arguments are a pointer to the device and EXCLUDE. The pointer is obtained through the *getdev* function.

EXCLUDE is defined in <mouse.h>.

### Diagnostics

---

This routine returns 0 if it succeeds. It returns -1 if there is no active event queue. It returns -2 if the *devinfo* argument does not point to a valid device. It returns -3 on attempts to exclude an excluded device, or attempts to reinclude an included device. It returns -4 if the action argument is invalid.

### See Also

---

*ev\_block*(S), *ev\_close*(S), *ev\_count*(S), *ev\_flush*(S), *ev\_getdev*(S), *ev\_getemask*(S), *ev\_init*(S), *ev\_open*(S), *ev\_pop*(S), *ev\_read*(S), *ev\_resume*(S), *ev\_setemask*(S), *ev\_suspend*(S)

### Notes

---

This routine must be linked in with the **-levent** linker option.

## ev\_getemask

---

return the current event mask

### Syntax

---

```
#include <types.h>
#include <param.h>
#include <sysmacros.h>
#include <page.h>
#include <sys/event.h>
#include <mouse.h>
```

```
int ev_getemask(emaskp)
emask_t *emaskp;
```

### Description

---

*ev\_getemask* returns the current event mask. This call takes a pointer to an event mask which is filled in. The program must have already opened an event queue.

This call complements *ev\_setemask*. The manual page for *ev\_setemask* describes an event mask in detail.

### Diagnostics

---

*ev\_getemask* returns -1 if there is no open event queue. Otherwise it returns 0.

### See Also

---

*ev\_block(S)*, *ev\_close(S)*, *ev\_count(S)*, *ev\_flush(S)*, *ev\_getdev(S)*, *ev\_gindev(S)*, *ev\_init(S)*, *ev\_open(S)*, *ev\_pop(S)*, *ev\_read(S)*, *ev\_resume(S)*, *ev\_setemask(S)*, *ev\_suspend(S)*

### Notes

---

This routine must be linked in with the **-levent** linker option.



## ev\_init

invokes the event manager

---

### Syntax

---

```
#include <types.h>
#include <param.h>
#include <sysmacros.h>
#include <page.h>
#include <sys/event.h>
#include <mouse.h>
```

```
int ev_init()
```

### Description

---

*ev\_init* reads the system event-configuration files and initializes the event manager. It is the first of two steps a program follows to obtain an event queue. Devices like mice or the keyboard may be read through an event queue. When *ev\_init* is called, the configuration files are read and checked for syntax. If there is an error or inconsistency, *ev\_init* returns an error. After the event manager is initialized, *ev\_open* should be called to obtain an event queue.

### Diagnostics

---

*ev\_init* returns 0 if it succeeds in reading the data files and initializing an event queue. Otherwise it returns -1.

### See Also

---

*ev\_block*(S), *ev\_close*(S), *ev\_count*(S), *ev\_flush*(S), *ev\_getdev*(S), *ev\_getemask*(S), *ev\_gindev*(S), *ev\_open*(S), *ev\_pop*(S), *ev\_read*(S), *ev\_resume*(S), *ev\_setemask*(S), *ev\_suspend*(S)

### Notes

---

This routine must be linked in with the **-levent** linker option.

### Files

---

```
/usr/lib/event/devices
/usr/lib/event/ttys
```

## ev\_pop

---

pop the next event off the queue

### Syntax

---

```
#include <types.h>
#include <param.h>
#include <sysmacros.h>
#include <page.h>
#include <sys/event.h>
#include <mouse.h>
```

```
int ev_pop()
```

### Description

---

*ev\_pop* clears the next event off the queue and returns the number of events lost due to queue overrun since the last *ev\_pop* call. An event queue must have been opened with *ev\_init(S)* and *ev\_open(S)*.

After an application is done with an event, the event is pop'ed off the queue. The queue is of fixed size, so if events are not pop'ed fast enough some might be lost due to overrun. A counter maintains the number of lost events. When *ev\_pop* is called, it clears the top event off of the queue, clears the counter and returns the number of lost events. This should always be zero, unless a program stops reading its event queue. If the queue is empty, *ev\_pop* returns -1.

When *ev\_pop* is called, the most recent pointer returned by *ev\_read* must be considered invalid, since that storage may be overwritten by the event driver.

### Diagnostics

---

*ev\_pop* returns -1 if there is not an open event queue. It returns -2 if there is nothing to pop because the queue is empty.

### See Also

---

*ev\_block(S)*, *ev\_close(S)*, *ev\_count(S)*, *ev\_flush(S)*, *ev\_getdev(S)*, *ev\_getemask(S)*, *ev\_gindev(S)*, *ev\_init(S)*, *ev\_open(S)*, *ev\_read(S)*, *ev\_resume(S)*, *ev\_setemask(S)*, *ev\_suspend(S)*

**Notes**

---

This routine must be linked in with the **-levent** linker option.



## **ev\_read**

---

read the next event in the queue

### **Syntax**

---

```
#include <types.h>
#include <param.h>
#include <sysmacros.h>
#include <page.h>
#include <sys/event.h>
#include <mouse.h>
```

```
EVENT* ev_read()
```

### **Description**

---

*ev\_read* returns a pointer to the next event in the queue or NULL if the queue is empty. Multiple calls to this routine return the same pointer until *ev\_pop* is called.

It is an error to call this routine from a program which does not have an open event queue.

### **Diagnostics**

---

This routine returns NULL if there is no event to read OR if there is not an open event queue.

### **See Also**

---

*ev\_block*(S), *ev\_close*(S), *ev\_count*(S), *ev\_flush*(S), *ev\_getdev*(S), *ev\_getemask*(S), *ev\_gindev*(S), *ev\_init*(S), *ev\_open*(S), *ev\_pop*(S), *ev\_resume*(S), *ev\_setemask*(S), *ev\_suspend*(S)

### **Notes**

---

This routine must be linked in with the **-levent** linker option.

## ev\_resume

---

restart a suspended queue

### Syntax

---

```
#include <types.h>
#include <param.h>
#include <sysmacros.h>
#include <page.h>
#include <sys/event.h>
#include <mouse.h>
```

```
int ev_resume()
```

### Description

---

*ev\_resume* restarts an event queue suspended by an *ev\_suspend* call.

### Diagnostics

---

This routine returns -1 to a program which does not have an open event queue. It returns -2 if the queue is not suspended. Normally it returns 0.

### See Also

---

*ev\_block(S)*, *ev\_close(S)*, *ev\_count(S)*, *ev\_flush(S)*, *ev\_getdev(S)*, *ev\_getemask(S)*, *ev\_gindev(S)*, *ev\_init(S)*, *ev\_open(S)*, *ev\_pop(S)*, *ev\_read(S)*, *ev\_setemask(S)*, *ev\_suspend(S)*

### Notes

---

This routine must be linked in with the **-levent** linker option.

## ev\_setemask

---

sets event mask

### Syntax

---

```
#include <types.h>
#include <param.h>
#include <sysmacros.h>
#include <page.h>
#include <sys/event.h>
#include <mouse.h>
```

```
int ev_setemask(emask)
emask_t emask;
```

### Description

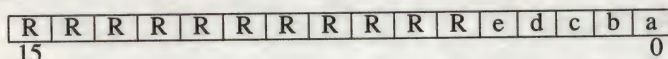
---

*ev\_setemask* sets the event mask on an event queue. Events whose tag is not masked in are prevented from entering an event queue. Event masks are always initialized to allow all events.

The different types of events are:

- R Reserved
- a "Other" Device events
- b Button events
- c String events
- d Relative Locator movement events
- e Absolute Locator movement events

The bits that make up the mask number have the following definitions:



### Diagnostics

---

If there is no open event queue -1 is returned. If the new event mask would cause no events to enter the queue, -2 is returned and the event mask is not changed. For example, on a queue with only a mouse, any event mask which did not include **D\_REL** would not allow any events to be enqueued.



**See Also**

---

ev\_block(S), ev\_close(S), ev\_count(S), ev\_flush(S), ev\_getdev(S),  
ev\_getemask(S), ev\_gindev(S), ev\_init(S), ev\_open(S), ev\_pop(S),  
ev\_read(S), ev\_resume(S), ev\_suspend(S)

## ev\_suspend

---

suspends an event queue.

### Syntax

---

```
#include <types.h>
#include <param.h>
#include <sysmacros.h>
#include <page.h>
#include <sys/event.h>
#include <mouse.h>
```

```
int ev_suspend()
```

### Description

---

*ev\_suspend* suspends a queue from receiving input. For example, if an application wants to fork a subshell, a call to *ev\_suspend* can suspend events until the subshell returns and the queue is resumed with an *ev\_resume* call. That way a process in the subshell can also have an event queue. This is required because the event manager only allows one active event queue per terminal or multiscreen.

### Diagnostics

---

This function returns -1 if no event queue is opened. It returns -2 if the queue is already suspended. Normally it returns zero.

### See Also

---

*ev\_block(S)*, *ev\_close(S)*, *ev\_count(S)*, *ev\_flush(S)*, *ev\_getdev(S)*, *ev\_getemask(S)*, *ev\_gindev(S)*, *ev\_init(S)*, *ev\_open(S)*, *ev\_pop(S)*, *ev\_read(S)*, *ev\_resume(S)*, *ev\_setemask(S)*

## **exec: execl, execv, execl, execve, execlp, execvp**

---

execute a file

### **Syntax**

---

```
int execl (path, arg0, arg1, ..., argn, (char *)0)
char *path, *arg0, *arg1, ..., *argn;
```

```
int execv (path, argv)
char *path, *argv[];
```

```
int execl (path, arg0, arg1, ..., argn, (char *)0, envp)
char *path, *arg0, *arg1, ..., *argn, *envp[];
```

```
int execve (path, argv, envp)
char *path, *argv[], *envp[];
```

```
int execlp (file, arg0, arg1, ..., argn, (char *)0)
char *file, *arg0, *arg1, ..., *argn;
```

```
int execvp (file, argv)
char *file, *argv[];
```

### **Description**

---

The *exec* system call in all its forms transforms the calling process into a new process. The new process is constructed from an ordinary, executable file called the *new process file*. This file consists of a header (see *a.out(F)*), a text segment, and a data segment. The data segment contains an initialized portion and an uninitialized portion (bss). There can be no return from a successful *exec* because the calling process is overlaid by the new process.

When a program is executed, it is called as follows:

```
main (argc, argv, envp)
int argc;
char **argv, **envp;
```

where *argc* is the argument count, *argv* is an array of character pointers to the arguments themselves, and *envp* is an array of character pointers to the environment strings. As indicated, *argc* is conventionally at least one, and the first member of the array points to a string containing the name of the file.



The *path* argument points to a path name that identifies the new process file.

The *file* argument points to the new process file. The path prefix for this file is obtained by a search of the directories passed as the *environment* line "PATH =" (see *environ*(M)). The environment is supplied by the shell (see *sh*(C)).

*arg0*, *arg1*, ..., *argn* are pointers to null-terminated character strings. These strings constitute the argument list available to the new process. By convention, at least *arg0* must be present and point to a string that is the same as *path* (or its last component).

*argv* is an array of character pointers to null-terminated strings. These strings constitute the argument list available to the new process. By convention, *argv* must have at least one member, and it must point to a string that is the same as *path* (or its last component). *argv* is terminated by a null pointer.

*envp* is an array of character pointers to null-terminated strings. These strings constitute the environment for the new process. *envp* is terminated by a null pointer. For *execl* and *execv*, the C run-time start-off routine places a pointer to the environment of the calling process in the global cell:

```
extern char **environ;
```

and it is used to pass the environment of the calling process to the new process.

File descriptors open in the calling process remain open in the new process, except for those whose close-on-exec flag is set; see *fcntl*(S). For those file descriptors that remain open, the file pointer is unchanged.

Signals set to terminate the calling process will be set to terminate the new process. Signals set to be ignored by the calling process will be set to be ignored by the new process. Signals set to be caught by the calling process will be set to terminate the new process; see *signal*(S).

For signals set by *sigset*(S), *exec* will ensure that the new process has the same system signal action for each signal type whose action is SIG\_DFL, SIG\_IGN, or SIG\_HOLD as the calling process. However, if the action is to catch the signal, then the action will be reset to SIG\_DFL, and any pending signal for this type will be held.

If the set-user-ID mode bit of the new process file is set (see *chmod*(S)), *exec* sets the effective user ID of the new process to the owner ID of the new process file. Similarly, if the set-group-ID mode

bit of the new process file is set, the effective group ID of the new process is set to the group ID of the new process file. The real user ID and real group ID of the new process remain the same as those of the calling process.

The shared memory segments attached to the calling process will not be attached to the new process (see *shmop*(S)).

Profiling is disabled for the new process; see *profil*(S).

The new process also inherits the following attributes from the calling process:

- nice value (see *nice*(S))
- process ID
- parent process ID
- process group ID
- semadj values (see *semop*(S))
- tty group ID (see *exit*(S) and *signal*(S))
- trace flag (see *ptrace*(S) request 0)
- time left until an alarm clock signal (see *alarm*(S))
- current working directory
- root directory
- file mode creation mask (see *umask*(S))
- file size limit (see *ulimit*(S))
- utime*, *stime*, *cutime*, and *cstime* (see *times*(S))
- file-locks (see *fcntl*(S) and *lockf*(S))

The *exec* system call will fail and return to the calling process if one or more of the following is true:

- |           |                                                                                                                                                                      |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [ENOENT]  | One or more components of the new process path name of the file do not exist.                                                                                        |
| [ENOTDIR] | A component of the new process path of the file prefix is not a directory.                                                                                           |
| [EACCES]  | Search permission is denied for a directory listed in the new process file's path prefix.                                                                            |
| [EACCES]  | The new process file is not an ordinary file.                                                                                                                        |
| [EACCES]  | The new process file mode denies execution permission.                                                                                                               |
| [ENOEXEC] | The <i>exec</i> is not an <i>execvp</i> or <i>execvp</i> , and the new process file has the appropriate access permission but an invalid magic number in its header. |
| [ETXTBSY] | The new process file is a pure procedure (shared text) file that is currently open for writing by some process.                                                      |



|             |                                                                                                                |
|-------------|----------------------------------------------------------------------------------------------------------------|
| [ENOMEM]    | The new process requires more memory than is allowed by the system-imposed maximum MAX-MEM.                    |
| [E2BIG]     | The number of bytes in the new process's argument list is greater than the system-imposed limit of 5120 bytes. |
| [EFAULT]    | Required hardware is not present.                                                                              |
| [EFAULT]    | <i>path</i> , <i>argv</i> , or <i>envp</i> point to an illegal address.                                        |
| [EAGAIN]    | Not enough memory.                                                                                             |
| [ELIBACC]   | Required shared library does not have execute permission.                                                      |
| [ELIBEXEC]  | Trying to <i>exec</i> (S) a shared library directly.                                                           |
| [EINTR]     | A signal was caught during the <i>exec</i> system call.                                                        |
| [ENOLINK]   | <i>path</i> points to a remote machine and the link to that machine is no longer active.                       |
| [EMULTIHOP] | Components of <i>path</i> require hopping to multiple remote machines.                                         |

## See Also

---

alarm(S), exit(S), fcntl(S), fork(S), nice(S), ptrace(S), semop(S), signal(S), sigset(S), times(S), ulimit(S), umask(S), lockf(S), a.out(F), environ(M), sh(C)

## Diagnostics

---

If *exec* returns to the calling process, an error has occurred; the return value will be -1 and *errno* will be set to indicate the error.

## Standards Conformance

---

*execl*, *execle*, *execlp*, *execv*, *execve* and *execvp* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
 The X/Open Portability Guide II of January 1987;  
 IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
 and NIST FIPS 151-1.



## execseg

---

makes a data region executable.

### Syntax

---

```
#include <xdata.h>
```

```
excode_t execseg(oldaddr, size)
exdata_t oldaddr;
unsigned size;
```

```
int unexecseg(addr)
excode_t addr;
```

### Description

---

*execseg(S)* is passed the current data address and size of the region to be executed and it returns the starting address of a region that is at least *size* number of bytes which can safely be branched to. On the Intel 8086 and 80286, processor an alias CS descriptor is associated with the same memory as the data segment in which the *oldaddr* region lies. This means that offsets in the executable segment to access a given byte are essentially the same as the offsets in the original data segment, except the selector is different.

Note that “excode\_t” and “exdata\_t” are “far” pointers on the 8086 and 80286 and segment selectors on the 386. On an architecture where pages in the same “segment” are any combination of read/write/execute, the returned address is identical to the parameter passed to *execseg(S)*.

We recommend that programs using this function on 8086- and 80286-based processors be large model, or that programmers be very familiar with “hybrid model” as well as with the use and misuse of far data.

When an error occurs, *execseg(S)* returns ((excode\_t)-1), with *errno* set to ENONEM. Errors include an invalid data address or *size*, and an inability to allocate a new data selector.

The *unexecseg()* system call disables an *addr* previously returned from *execseg(S)* from being used as an executable region. Specifically, on the 8086 and 80286 architectures, this call frees the selector used for the executable region. It returns 0 on success, or a -1 on error. For example, if *addr* is not an address returned by *execseg(S)*, then a -1 is returned and it can be used as an executable region.

## Example

---

```

excode_t funcp; char far *datap;
.
.
.
datap=brkctl(BR_NEWSEG,1000L,0L);

/* load executable code into
 data region datap */
load_with_code(datap,1000)

funcp=execseg(datap,1000); (*funcp)()

/*call subroutine*/ if (unexecseg (funcp)==-1){
 printf("unexecseg failed\n"); exit(1); }

```

## Notes

---

On the Intel 8086 and 80286 architectures, *execseg*(S) expects far addresses to be passed. Only experienced programmers should use this feature.

Since the *execseg* return value and address arguments are "far" pointers on 86 and 286 machines, any program including *xdata.h* must be compiled using the **-Me** option.

The following restrictions apply to the execute data system call.

1. Even though an address and size are passed to *execseg*, the entire segment containing the requested addresses are aliased. The address and size are validated before the aliasing is allowed.
2. No part of the data segment that is aliased may be deallocated (via *sbrk*(S) or *brkctl*(S)) while it is aliased. This restriction applies to the entire segment that is aliased, even if only a small piece of the segment was aliased. After *unexecseg*ing the aliased segment, the data segment may be deallocated.
3. Each call to *execseg* results in a new alias segment being used, even if the data segment is already aliased.

Due to compiler confusion, you may get the message "at least one void operand" when using *execseg*. Please ignore it.

Programs using this call must be compiled with the **-lx** option.

Under UNIX-386 the *oldaddr* parameter passed to *execseg*() is a segment selector, not a pointer. *execseg*() returns a selector value which provides a code segment alias to the original segment.

Similarly, the *addr* parameter passed to *unexecseg()* is a selector value that has previously been returned by *execseg()*.

Note that since *execseg()* returns a segment selector on 386 machines, the return value from *execseg()* is not directly useable in a C program. In an assembler program this selector value would be used as the top 16 bits of a 48 bit far pointer.



## exit, \_exit

---

terminate process

### Syntax

---

```
void exit (status)
int status;
void _exit (status)
int status;
```

### Description

---

The *exit* system call terminates the calling process with the following consequences:

All of the file descriptors open in the calling process are closed.

If the parent process of the calling process is executing a *wait*, it is notified of the calling process's termination and the low order eight bits (that is, bits 0377) of *status* are made available to it (see *wait(S)*).

If the parent process of the calling process is not executing a *wait*, the calling process is transformed into a zombie process. A *zombie process* is a process that only occupies a slot in the process table. It has no other space allocated either in user or kernel space. The process table slot that it occupies is partially overlaid with time accounting information (see *<sys/proc.h>*) to be used by *times*.

The parent process ID of all of the calling processes' existing child processes and zombie processes is set to 1. This means the initialization process (see *intro(S)*) inherits each of these processes.

Each attached shared memory segment is detached and the value of *shm\_nattach* in the data structure associated with its shared memory identifier is decremented by 1.

For each semaphore for which the calling process has set a *semadj* value (see *semop(S)*), that *semadj* value is added to the *semval* of the specified semaphore.

If the process has a process, text, or data lock, an *unlock* is performed (see *plock(S)*).

An accounting record is written on the accounting file if the system's accounting routine is enabled (see *acct(S)*).

If the process ID, tty group ID, and process group ID of the calling process are equal, the **SIGHUP** signal is sent to each process that has a process group ID equal to that of the calling process.

A death of child signal is sent to the parent.

The C function *exit* may cause cleanup actions before the process exits. The function *\_exit* circumvents all cleanup.

## See Also

---

acct(S), intro(S), plock(S), semop(S), signal(S), sigset(S), wait(S)

## Diagnostics

---

None. There can be no return from an *exit* system call.

## Standards Conformance

---

*\_exit* is conformant with:

The X/Open Portability Guide II of January 1987;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

*exit* is conformant with:

The X/Open Portability Guide II of January 1987;  
ANSI X3.159-198X C Language Draft Standard, May 13, 1988;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

## **exp, log, log10, pow, sqrt**

exponential, logarithm, power, square root functions

### **Syntax**

```
#include <math.h>
```

```
double exp (x)
double x;
```

```
double log (x)
double x;
```

```
double log10 (x)
double x;
```

```
double pow (x, y)
double x, y;
```

```
double sqrt (x)
double x;
```

### **Description**

The *exp* function returns  $e^x$ .

*log* returns the natural logarithm of  $x$ . The value of  $x$  must be positive.

*log10* returns the logarithm base ten of  $x$ . The value of  $x$  must be positive.

*pow* returns  $x^y$ . If  $x$  is zero,  $y$  must be positive. If  $x$  is negative,  $y$  must be an integer.

*sqrt* returns the non-negative square root of  $x$ . The value of  $x$  may not be negative.

### **See Also**

hypot(S), matherr(S), sinh(S)



## Diagnostics

---

The *exp* function returns **HUGE** when the correct value would overflow, or 0 when the correct value would underflow, and sets *errno* to **ERANGE**.

*log* and *log10* return **-HUGE** and set *errno* to **EDOM** when *x* is non-positive. A message indicating DOMAIN error (or SING error when *x* is 0) is printed on the standard error output.

*pow* returns 0 and sets *errno* to **EDOM** when *x* is 0 and *y* is non-positive, or when *x* is negative and *y* is not an integer. In these cases a message indicating DOMAIN error is printed on the standard error output. When the correct value for *pow* would overflow or underflow, *pow* returns **±HUGE** or 0 respectively, and sets *errno* to **ERANGE**.

*sqrt* returns 0 and sets *errno* to **EDOM** when *x* is negative. A message indicating DOMAIN error is printed on the standard error output.

These error-handling procedures may be changed with the function *matherr*(S).

## Standards Conformance

---

*exp* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

*log*, *log10*, *pow* and *sqrt* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
ANSI X3.159-198X C Language Draft Standard, May 13, 1988;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

# **fclose, fflush**

---

close or flush a stream

## **Syntax**

---

**#include** <stdio.h>

**int** fclose (stream)  
**FILE** \*stream;

**int** fflush (stream)  
**FILE** \*stream;

## **Description**

---

The *fclose* function causes any buffered data for the named *stream* to be written out, and the *stream* to be closed.

The *fclose* function is performed automatically for all open files upon calling *exit*(S).

*fflush* causes any buffered data for the named *stream* to be written to that file. The *stream* remains open.

## **See Also**

---

close(S), exit(S), fopen(S), setbuf(S), stdio(S)

## **Diagnostics**

---

These functions return 0 for success and EOF if any error (such as trying to write to a file that has not been opened for writing) was detected.

## **Standards Conformance**

---

*fclose* and *fflush* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
ANSI X3.159-198X C Language Draft Standard, May 13, 1988;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

# fcntl

## file control

---

### Syntax

---

```
#include <fcntl.h>
```

```
int fcntl (fildes, cmd, arg)
int fildes, cmd;
```

### Description

---

The *fcntl* system call provides for control over open files. The *fildes* argument is an open file descriptor obtained from a *creat*, *open*, *dup*, *fcntl*, or *pipe* system call. The data type and value of *arg* are specific to the type of command specified by *cmd*. The symbolic names for commands and file status flags are defined by the <fcntl.h> header file.

The commands available are:

**F\_DUPFD** Return a new file descriptor as follows:

Lowest numbered available file descriptor greater than or equal to *arg*.

Same open file (or pipe) as the original file.

Same file pointer as the original file (that is, both file descriptors share one file pointer).

Same access mode (read, write, or read/write).

Same file status flags (that is, both file descriptors share the same file status flags).

The close-on-exec flag associated with the new file descriptor is set to remain open across *exec*(S) system calls.

**F\_GETFD** Get the close-on-exec flag associated with the file descriptor *fildes*. If the low-order bit is **0**, the file will remain open across *exec*; otherwise the file will be closed upon execution of *exec*.

**F\_SETFD** Set the close-on-exec flag associated with *fildes* to the low-order bit of *arg* (**0** or **1** as above).



**F\_GETFL**     Get *file* status flags (see *open(S)*).

**F\_SETFL**     Set *file* status flags to *arg*. Only certain flags can be set (see *fcntl(M)*).

The following commands are used for file-locking and record-locking. Locks may be placed on an entire file or segments of a file.

#### **F\_GETLK**

Get the first lock that blocks the lock description given by the variable of type *struct flock* pointed to by *arg*. The information retrieved overwrites the information passed to *fcntl* in the *flock* structure. If no lock is found that would prevent this lock from being created, then the structure is passed back unchanged except for the lock type which will be set to **F\_UNLCK**.

#### **F\_SETLK**

Set or clear a file segment lock. According to the variable of type *struct flock* pointed to by *arg* (see *fcntl(M)*), the *cmd* **F\_SETLK** is used to establish read (**F\_RDLCK**) and write (**F\_WRLCK**) locks. It also is used to remove either type of lock (**F\_UNLCK**). If a read or write lock cannot be set, *fcntl* returns immediately with an error value of -1.

#### **F\_SETLKW**

This *cmd* is the same as **F\_SETLK** except that if a read or write lock is blocked by other locks, the process will sleep until the segment is free to be locked.

A read lock prevents any process from write locking the protected area. More than one read lock may exist for a given segment of a file at a given time. The file descriptor on which a read lock is being placed must have been opened with read access.

A write lock prevents any process from read-locking or write-locking the protected area. Only one write lock may exist for a given segment of a file at a given time. The file descriptor on which a write lock is being placed must have been opened with write access.

The structure *flock* defined in the *<fcntl.h>* header file describes a lock. It describes the type (*l\_type*), starting offset (*l\_whence*), relative offset (*l\_start*), size (*l\_len*), and process-ID (*l\_pid*):

```
short l_type; /* F_RDLCK, F_WRLCK, F_UNLCK */
short l_whence; /* flag for starting offset */
long l_start; /* relative offset in bytes */
long l_len; /* if 0 then until EOF */
short l_pid; /* returned with F_GETLK */
```

The value of *l\_whence* is 0, 1, or 2 to indicate that the relative offset, *l\_start* bytes, will be measured from the start of the file, current position, or end of file, respectively. The value of *l\_len* is the number of

consecutive bytes to be locked. The process id is used only with the `F_GETLK` *cmd* to return the values for a blocking lock. Locks may start and extend beyond the current end of a file, but may not be negative relative to the beginning of the file. A lock may be set to always extend to the end of file by setting *l\_len* to zero (0). If such a lock also has *l\_whence* and *l\_start* set to zero (0), the whole file will be locked. Changing or unlocking a segment from the middle of a larger locked segment leaves two smaller segments for either end. Locking a segment that is already locked by the calling process causes the old lock type to be removed and the new lock type to take effect. All locks associated with a file for a given process are removed when a file descriptor for that file is closed by that process or the process holding that file descriptor terminates. Locks are not inherited by a child process in a `fork(S)` system call.

When mandatory file and record locking is active on a file (see `chmod(S)`), *read* and *write* system calls issued on the file will be affected by the record locks in effect.

The `fcntl` system call will fail if one or more of the following is true:

- |          |                                                                                                                                                                                                                                                                                                                                                                 |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [EBADF]  | The <i>files</i> argument is not a valid open file descriptor.                                                                                                                                                                                                                                                                                                  |
| [EINVAL] | The <i>cmd</i> argument is <code>F_DUPFD</code> . The <i>arg</i> argument is either negative, or greater than or equal to the configured value for the maximum number of open file descriptors allowed each user.                                                                                                                                               |
| [EINVAL] | The <i>cmd</i> argument is <code>F_GETLK</code> , <code>F_SETLK</code> , or <code>SETLKW</code> and <i>arg</i> or the data it points to is not valid.                                                                                                                                                                                                           |
| [EACCES] | The <i>cmd</i> argument is <code>F_SETLK</code> , the type of lock ( <i>l_type</i> ) is a read ( <code>F_RDLCK</code> ) lock, and the segment of a file to be locked is already write locked by another process or the type is a write ( <code>F_WRLCK</code> ) lock and the segment of a file to be locked is already read or write locked by another process. |
| [ENOLCK] | The <i>cmd</i> argument is <code>F_SETLK</code> or <code>F_SETLKW</code> , the type of lock is a read or write lock, and there are no more record locks available (too many file segments locked) because the system maximum has been exceeded.                                                                                                                 |
| [EMFILE] | The <i>cmd</i> argument is <code>F_DUPFD</code> and file-descriptors are currently open in the calling-process.                                                                                                                                                                                                                                                 |
| [EBADF]  | The <i>cmd</i> argument is <code>F_SETLK</code> or <code>F_SETLKW</code> , the type of lock ( <i>l_type</i> ) is a read-lock ( <code>F_RDLCK</code> ), and <i>files</i> is not a valid file-descriptor open for reading.                                                                                                                                        |



- [EBADF] The *cmd* argument is F\_SETLK or F\_SETLKW, the type of lock (*l\_type*) is a write-lock (F\_WRLCK), and *fildev* is not a valid file-descriptor open for writing.
- [EDEADLK] The *cmd* argument is F\_SETLKW, the lock is blocked by some lock from another process, and putting the calling-process to sleep, waiting for that lock to become free, would cause a deadlock.
- [EFAULT] The *cmd* argument is F\_SETLK, *arg* points outside the program address space.
- [EINTR] A signal was caught during the *fcntl* system call.
- [ENOLINK] *fildev* is on a remote machine and the link to that machine is no longer active.

## See Also

---

close(S), creat(S), dup(S), exec(S), fork(S), open(S), pipe(S), fcntl(M)

## Diagnostics

---

Upon successful completion, the value returned depends on *cmd* as follows:

- |          |                                                    |
|----------|----------------------------------------------------|
| F_DUPFD  | A new file descriptor.                             |
| F_GETFD  | Value of flag (only the low-order bit is defined). |
| F_SETFD  | Value other than -1.                               |
| F_GETFL  | Value of file flags.                               |
| F_SETFL  | Value other than -1.                               |
| F_GETLK  | Value other than -1.                               |
| F_SETLK  | Value other than -1.                               |
| F_SETLKW | Value other than -1.                               |

Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## Warnings

---

Because in the future the variable *errno* will be set to EAGAIN rather than EACCES when a section of a file is already locked by another process, portable application programs should expect and test for either value.



## **Standards Conformance**

---

*fcntl* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;  
and NIST FIPS 151-1.

## **error, feof, clearerr, fileno**

---

stream status inquiries

### **Syntax**

---

**#include <stdio.h>**

**int error (stream)**  
**FILE \*stream;**

**int feof (stream)**  
**FILE \*stream;**

**void clearerr (stream)**  
**FILE \*stream;**

**int fileno (stream)**  
**FILE \*stream;**

### **Description**

---

The *error* function returns non-zero when an I/O error has previously occurred reading from or writing to the named *stream*, otherwise zero.

*feof* returns non-zero when EOF has previously been detected reading the named input *stream*, otherwise zero.

*clearerr* resets the error indicator and EOF indicator to zero on the named *stream*.

*fileno* returns the integer file descriptor associated with the named *stream*; see *open*(S).

### **See Also**

---

*open*(S), *fopen*(S), *stdio*(S)

### **Notes**

---

All these functions are implemented as macros; they cannot be declared or redeclared.

## Standards Conformance

---

*clearerr*, *feof* and *ferror* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

*fileno* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.



## fgetpos

---

gets and stores the current value of a stream's file position indicator

### Syntax

---

```
#include <stdio.h>

int fgetpos(stream, pos)
FILE *stream;
fpos_t *pos;
```

### Description

---

The *fgetpos* function gets the current value of *stream*'s file position indicator and stores it in the object that *pos* points to. The *fsetpos* function can later use information stored in *pos* to reset *stream*'s pointer to its position at the time *fgetpos* was called.

### Notes

---

The *pos* value is stored in an internal format and is intended for use only by the *fgetpos* and *fsetpos* functions.

### Return Value

---

If successful, the *fgetpos* function returns 0. On failure, it returns a nonzero value and sets *errno* to one of the following manifest constants (defined in *stdio.h*):

| Constant | Meaning                                                               |
|----------|-----------------------------------------------------------------------|
| EINVAL   | The <i>stream</i> value is invalid.                                   |
| EBADF    | The specified stream is not a valid file handle or is not accessible. |

### See Also

---

fsetpos(S)

## Example

---

```
#include <stdio.h>

FILE *stream;
fpos_t pos[];
int val, i;
char fcontents[13]; /* buffer for full contents of file1 */
char writein1[14]= "Hello world!0;
char writein2[8]= "there.0;

main()
{
 if ((stream = fopen("file1","w+")) == NULL) /* open file1 */
 {
 perror("fopen");
 printf ("Trouble opening file\n");
 }
 else /* write in 1st string */
 fwrite(writein1,sizeof(char),13,stream);

 position=0;
 fsetpos(stream, &position); /*reset file to beginning */

 fread(fcontents, sizeof(char), 13, stream); /* read contents */
 printf("%s", fcontents); /* and print it */

 position=0;
 fsetpos(stream, &position); /*reset file to beginning */

 fread(fcontents, sizeof(char), 5, stream); /* read "Hello" */
 if (fgetpos(stream, &position) != 0) /* Save current position */
 perror("fgetpos error");

 fread(fcontents, sizeof(char), 8, stream); /* read some more */
 if (fsetpos(stream, &position) != 0) /* Return to saved position */
 perror("fsetpos error");

 /* Overwrite file from this position with the word "there" */

 fwrite(writein2, sizeof(char), 8, stream);

 position=0;
 fsetpos(stream, &position); /*reset file to beginning */
 fread(fcontents, sizeof(char), 13, stream);
 printf("%s", fcontents); /* print contents */
}
}
```

This program opens a file named *file1* and prints "Hello world!" It then outputs the file contents on screen. Next, it reads the first five characters in the file *hello* and calls *fgetpos* to find and save the file position pointer. After performing another read, the program calls *fsetpos* to restore the file pointer to the saved position.

## **Standards Conformance**

---

*fgetpos* is conformant with:

ANSI X3.159-198X C Language Draft Standard, May 13, 1988.



## field

---

### FIELD library routines

### Syntax

---

```
#include <form.h>
```

```
cc [flags] files -lform -lcurses [libraries]
```

```
FIELD * new_field (r, c, frow, fcol, nrow, nbuf)
int r, c, frow, fcol, nrow, nbuf;
```

```
FIELD * dup_field (field, frow, fcol)
FIELD * field;
int frow, fcol;
```

```
FIELD * link_field (field, frow, fcol)
FIELD * field;
int frow, fcol;
```

```
int free_field (field)
FIELD * field;
```

```
int field_info (field, rows, cols, frow, fcol, nrow, nbuf)
FIELD * field;
int * rows, * cols, * frow, * fcol, * nrow, nbuf;
```

```
int move_field (field, frow, fcol)
FIELD * field;
int frow, fcol;
```

```
int set_field_type (field, type, [arg_1, arg_2, ...])
FIELD * field;
FIELDTYPE * type;
```

```
FIELDTYPE * field_type (field)
FIELD * field;
```

```
char * field_arg (field)
FIELD * field;
```

```
int set_field_just (field, justification)
FIELD * field;
int justification;
```

```
int field_just (field)
FIELD * field;
```

```
int set field fore (field, fore)
FIELD * field;
int fore;
```

```
int field fore (field)
FIELD * field;
```

```
int set field back (field, back)
FIELD * field;
int back;
```

```
int field back (field)
FIELD * field;
```

```
int set field pad (field, pad)
FIELD * field;
int pad;
```

```
int field pad (field)
FIELD * field;
```

```
int set field buffer (field, buf, value)
FIELD * field;
int buf;
char * value;
```

```
char * field buffer (field, buf)
FIELD * field;
int buf;
```

```
int set field status (field, status)
FIELD * field;
int status;
```

```
int field status (field)
FIELD * field;
```

```
int set field userptr (field, userptr)
FIELD * field;
char * userptr;
```

```
char * field userptr (field)
FIELD * field;
```

```
int set field opts (field, opts)
FIELD * field;
OPTIONS opts;
```

```
int field opts on (field, opts)
FIELD * field;
OPTIONS opts;
```

```
int field_opts_off (field, opts)
FIELD * field;
OPTIONS opts;
```

```
OPTIONS field_opts (field)
FIELD * field;
```

options:

```
O_ACTIVE
O_PUBLIC
O_EDIT
O_WRAP
O_BLANK
O_AUTOSKIP
O_NULLOK
```

## Description

---

These FIELD routines run on any terminal supported by *curses*(S), the low-level ETI library. Once you compile your UNIX program **#include**ing the header file **form.h**, you should link it with the **form** and **curses** library routines.

## Functions

---

The following is a list of FIELD routines. For a complete description of each routine, see the *Programmer's Guide*.

*new\_field(r,c, frow, fcol, nrow, nbuf)* creates a new field with *r* rows, *c* columns; starting at *frow, fcol* in the subwindow of the form to contain the field; with *nrow* offscreen rows and *nbuf* additional work buffers. It returns a pointer to the created field. In general, you should store these field pointers in an array.

*dup\_field(field, frow, fcol)* duplicates the given field at the named location.

*link\_field(field, frow, fcol)* also duplicates the given field at the named location. However, unlike *dup\_field()* it shares the field buffers between both occurrences of the field and permits the setting of different attributes for each field.

*free\_field(field)* frees the storage allocated for the given field.

*field\_info(field, rows, cols, frow, fcol, nrow, nbuf)* returns the size, position, and other named field characteristics to the locations pointed to by the pointer arguments *rows, cols, frow, fcol, nrow*, and *nbuf*.



*move\_field(field,frow, fcol)* moves the disconnected **field** to the location **frow**, **fcol** in the form subwindow.

*set\_field\_type(field,type, arg\_1,arg\_2,...)* associates the given field type with *field*. Certain field types take additional arguments. **TYPE\_ALNUM**, for instance, requires one, the minimum width specification for the field.

*field\_type(field)* returns a pointer to the field type of *field*.

*field\_arg(field)* returns a pointer to the field arguments associated with the field type of *field*.

*set\_field\_just(field,justification)* sets the justification for the given field. Justification may be **NO\_JUSTIFICATION**, **JUSTIFY\_RIGHT**, **JUSTIFY\_LEFT**, or **JUSTIFY\_CENTER**.

*field\_just(field)* returns the indicator of the justification for the field.

*set\_field\_fore(field,fore)* sets the foreground attribute of *field*. The foreground attribute is the low-level visual display attribute used to display the field characters.

*field\_fore(field)* returns the foreground attribute of *field*.

*set\_field\_back(field,back)* sets the background attribute of *field*. The background attribute is the low-level visual display attribute used to display the area immediately surrounding the field characters.

*field\_back(field)* returns the background attribute of *field*.

*set\_field\_pad(field,pad)* sets the pad (blank) character for *field*.

*field\_pad(field)* returns the pad character for *field*.

*set\_field\_buffer(field,buf, value)* sets buffer *buf* of *field* to *value*. Buffer 0 stores the displayed value of the field.

*field\_buffer(field,buf)* returns the value of *field* buffer *buf*.

Every field has an associated status flag that is set whenever the field's value (field buffer 0) changes. *set\_field\_status(field,status)* sets the field's status flag to *status*.

*field\_status(field)* returns the status of *field*.

Every field has an associated user pointer that you can use to store pertinent data.

*set\_field\_userptr(field,userptr)* sets the field's user pointer.

*field\_userptr(field)* returns the field's user pointer.

*set\_field\_opts(field,opts)* turns on the named options of the field and turns off all its remaining options. Options are boolean values.

*field\_opts\_on(field,opts)* turns on the named options.

*field\_opts\_off(field,opts)* turns off the named options.

*field\_opts(field)* returns the field's options setting. To set options, you can apply boolean operators to the value returned by *field\_opts()* and let the result be the second argument to *set\_field\_opts()*.

**options:**

|                   |                                                                                  |
|-------------------|----------------------------------------------------------------------------------|
| <b>O_VISIBLE</b>  | field displayed                                                                  |
| <b>O_ACTIVE</b>   | field visited during processing                                                  |
| <b>O_PUBLIC</b>   | field displayed as data entered                                                  |
| <b>O_EDIT</b>     | field can be edited                                                              |
| <b>O_WRAP</b>     | words not fitting on field line are wrapped to next line                         |
| <b>O_BLANK</b>    | whole field erased if first character entered before any other character changed |
| <b>O_AUTOSKIP</b> | moves to start of next field when current field full                             |
| <b>O_NULLOK</b>   | can leave blank field without validating it                                      |

## Diagnostics

---

The following values are returned by one or more routines that return an integer. For specific information on which routines return which value, see the *Programmer's Guide*.

|                        |                                   |
|------------------------|-----------------------------------|
| <b>E_OK</b>            | function returned successfully    |
| <b>E_CONNECTED</b>     | object is connected               |
| <b>E_SYSTEM_ERROR</b>  | system error                      |
| <b>E_BAD_ARGUMENT</b>  | argument is incorrect             |
| <b>E_CURRENT</b>       | field is current field            |
| <b>E_POSTED</b>        | form is posted                    |
| <b>E_INVALID_FIELD</b> | field is invalid                  |
| <b>E_NOT_CONNECTED</b> | object is not connected           |
| <b>E_NO_ROOM</b>       | form does not fit in subwindow    |
| <b>E_BAD_STATE</b>     | called from inappropriate routine |

FIELD (S)

FIELD (S)

**E\_UNKNOWN\_COMMAND** unknown command was given to the  
form driver

**E\_REQUEST\_DENIED** recognized request failed

## **See Also**

---

curses(S), fieldtype(S), form(S), item(S), menu(S), panel(S), tam(S)



## fields

return status based on fields of authentication database

### Syntax

---

```
#include <sys/types.h>
#include <sys/security.h>
#include <sys/audit.h>
#include <prot.h>
```

```
int running_secure ()
```

```
int locked_out (pr)
struct pr_passwd *pr;
```

```
void read_pw_fields (fld, flg)
struct pr_field *fld;
struct pr_flag *flg;
```

```
void store_pw_fields (f, name, fd, fg)
FILE *f;
char *name;
struct pr_field *fd;
struct pr_flag *fg;
```

```
void read_tc_fields (fld, flg)
struct t_field *fld;
struct t_flag *flg;
```

```
void store_tc_fields (f, name, fd, fg)
FILE *f;
char *name;
struct t_field *fd;
struct t_flag *fg;
```

```
void read_fi_fields (fld, flg)
struct f_field *fld;
struct f_flag *flg;
```

```
void store_fi_fields (f, name, fd, fg)
FILE *f;
char *name;
struct f_field *fd;
struct f_flag *fg;
```

```
void read_cm_fields (fld, flg)
struct c_field *fld;
struct c_flag *flg;
```

```
void store_cm_fields (f, name, fd, fg)
FILE *f;
char *name;
struct c_field *fd;
struct c_flag *fg;
```

```
void store_df_fields (f, name, pr)
register FILE *f;
char *name;
struct pr_default *pr;
```

## Description

---

These routines return various information based on the user, process environment, and the values within the authentication database.

*Running\_secure* returns a 0 if the system is not running in a secure state and non-zero if the system is running in a secure state. Programs can use this to determine if extra checking needs to be done or if tighter controls on data (e.g., less file permissions) need to be presented.

*Locked\_out* returns 1 if the user represented in the *pr\_passwd* structure referenced by *pr* cannot log in for some reason and returns 0 if the user is able to log in. The reasons include: unconditional lock-out from the system administrator, too many unsuccessful login tries, and/or the password lifetime has past. This routine is used prior to fashioning a session for an account, be it login, at, cron, or su sessions.

*Read\_pw\_fields* fills in the *pr\_field* and *pr\_flag* parts of a *pr\_passwd* structure with the contents from the current protected password entry. This entry must previously be obtained with a *getprpwent*(S) or *getprpwnam*(S) call.

*Store\_pw\_fields* is the inverse of *read\_pw\_fields*. It creates a file entry based on one of the *pr\_passwd* structure components *pr\_field* and *pr\_flag*, and writes them to the protected password database under the key *name* to file *f*.

The *read\_fi\_fields* and *store\_fi\_fields* routines, *read\_tc\_fields* and *store\_tc\_fields* routines, and *read\_cm\_fields* and *store\_cm\_fields* routines perform similar actions for the File Control, Terminal Control and Command Control Databases respectively, using the appropriate field and flag structures. perform similar actions for the *Store\_df\_fields* updates the default entry *name* in file *f* with information held in *pr*.

The *read\_pw\_fields*, *read\_fi\_fields*, *read\_tc\_fields* and *read\_cm\_fields* routines may be applied to either an entry in their respective databases or to a defaults entry. The **pr\_passwd**, **pr\_term**, **pr\_file** and **pr\_command** structures contain extra elements to hold default values.

## Notes

---

The non-zero binding of *running\_secure* and the type of secure class of operation (TCSEC C1, C2, B1, B2, B3, A1) is yet to be determined.

## See Also

---

getprpwent(S), getprtcent(S), getprfient(S), getprcment(S),  
getprdfent(S), authcap(S), authcap(F)



# fieldtype

---

## FIELDTYPE library routines

### Syntax

---

```
#include <form.h>
```

```
cc [flags] files -lform -lcurses [libraries]
```

```
typedef int (* PTF_int) ();
```

```
FIELDTYPE * new_fieldtype (field_check, char_check)
```

```
PTF_int field_check;
```

```
PTF_int char_check;
```

```
int free_fieldtype(fieldtype);
```

```
FIELDTYPE * fieldtype;
```

```
typedef char * (* PFT_charP) ();
```

```
typedef void (* PFT_void) ();
```

```
int set_fieldtype arg (fieldtype, mak_arg, cpy_arg, free_arg)
```

```
FIELDTYPE * fieldtype;
```

```
char * mak_arg(ap);
```

```
va_list * ap;
```

```
PTF_charP cpy_arg;
```

```
PTF_void free_arg;
```

```
typedef char * (* PFT_charP) ();
```

```
int set_fieldtype_choice (fieldtype, next_choice, prev_choice)
```

```
FIELDTYPE * fieldtype;
```

```
PTF_int next_choice;
```

```
PTF_int prev_choice;
```

```
int next_choice (FIELD * f, char * arg);
```

```
int prev_choice (FIELD * f, char * arg);
```

```
FIELDTYPE * link_fieldtyp (type1,type2)
```

```
FIELDTYPE * type1;
```

```
FIELDTYPE * type2;
```

### Description

---

These FIELDTYPE routines run on any terminal supported by *curses*(S), the low-level UNIX library. Once you compile your UNIX program **#include**ing the header file **form.h**, you should link it with the **form** and **curses** library routines.

## Functions

---

The following is a list of FIELDTYPE routines. For a complete description of each routine, see the *Programmer's Guide*.

*new\_fldtype* (*field\_check*, *char\_check*) creates a new field type. You must write functions *field\_check*, which validates the field value and *char\_check*, which validates each character.

*free\_fldtype* (*fieldtype*) frees the space allocated for the given field type.

By associating the given function pointers with the field type, *set\_fldtype\_arg* (*fieldtype*, *mak\_arg*, *cpy\_arg*, *free\_arg*) connects to the field type additional arguments necessary for a *set\_fldtype* ( ) call. Function *mak\_arg* allocates a structure for the field specific parameters to *set\_fldtype* ( ) and returns a pointer to the saved data. Function *copy\_arg* duplicates the structure created by *make\_arg*. Function *free\_arg* frees any storage allocated by *make\_arg* or *copy\_arg*.

Requests REQ\_NEXT\_CHOICE and REQ\_PREV\_CHOICE let the user choose the next or previous value of a field type comprising an ordered set of values. *set\_fldtype\_choice* (*fieldtype*, *next\_choice*, *prev\_choice*) enables you to implement these requests for the given field type. It associates with the given field type application-defined functions that return pointers to the next or previous choice for the field.

*link\_fldtype* (*type1*, *type2*) returns a pointer to the field type built from the two given types. The constituent types may be any application-defined or -defined types.

## See Also

---

curses(S), form(S), field(S), panel(S), menu(S), item(S), tam(S)

## Diagnostics

---

The following values are returned by one or more routines that return an integer. For specific information on which routines return which value, see the *Programmer's Guide*.

|                    |                                |
|--------------------|--------------------------------|
| <b>E_OK</b>        | function returned successfully |
| <b>E_CONNECTED</b> | object is connected            |

FIELDTYPE (S)

FIELDTYPE (S)

|                          |                                                 |
|--------------------------|-------------------------------------------------|
| <b>E_SYSTEM_ERROR</b>    | system error                                    |
| <b>E_BAD_ARGUMENT</b>    | argument is incorrect                           |
| <b>E_CURRENT</b>         | field is current field                          |
| <b>E_POSTED</b>          | form is posted                                  |
| <b>E_INVALID_FIELD</b>   | field is invalid                                |
| <b>E_NOT_CONNECTED</b>   | object is not connected                         |
| <b>E_NO_ROOM</b>         | form does not fit in subwindow                  |
| <b>E_BAD_STATE</b>       | called from inappropriate routine               |
| <b>E_UNKNOWN_COMMAND</b> | unknown command was given to the<br>form driver |
| <b>E_REQUEST_DENIED</b>  | recognized request failed                       |



## floor, ceil, fmod, fabs

floor, ceiling, remainder, absolute value functions

### Syntax

---

```
#include <math.h>
```

```
double floor (x)
double x;
```

```
double ceil (x)
double x;
```

```
double fmod (x, y)
double x, y;
```

```
double fabs (x)
double x;
```

### Description

---

*floor* returns the largest integer (as a double-precision number) not greater than  $x$ .

*ceil* returns the smallest integer not less than  $x$ .

*fmod* returns the floating-point remainder of the division of  $x$  by  $y$ :  $x$  if  $y$  is zero or if  $x/y$  would overflow; otherwise the number  $f$  with the same sign as  $x$ , such that  $x = iy + f$  for some integer  $i$ , and  $|f| < |y|$ .

*fabs* returns the absolute value of  $x$ ,  $|x|$ .

### See Also

---

abs(S)

### Standards Conformance

---

*ceil*, *fabs*, *floor* and *fmod* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

# **fopen, freopen, fdopen**

---

open a stream

## **Syntax**

---

**#include <stdio.h>**

**FILE \*fopen (filename, type)**  
**const char \*filename, \*type;**

**FILE \*freopen (filename, type, stream)**  
**const char \*filename, \*type;**  
**FILE \*stream;**

**FILE \*fdopen (fildes, type)**  
**int fildes;**  
**const char \*type;**

## **Description**

---

The *fopen* function opens the file named by *filename* and associates a *stream* with it. The *fopen* function returns a pointer to the FILE structure associated with the *stream*.

*filename* points to a character string that contains the name of the file to be opened.

*type* is a character string having one of the following values:

|      |                                                                |
|------|----------------------------------------------------------------|
| "r"  | open for reading                                               |
| "w"  | truncate or create for writing                                 |
| "a"  | append; open for writing at end of file, or create for writing |
| "r+" | open for update (reading and writing)                          |
| "w+" | truncate or create for update                                  |
| "a+" | append; open or create for update at end-of-file               |

*freopen* substitutes the named file in place of the open *stream*. The original *stream* is closed, regardless of whether the open ultimately succeeds. *freopen* returns a pointer to the FILE structure associated with *stream*.

# fork

---

create a new process

## Syntax

---

`int fork ()`

## Description

---

The *fork* system call causes creation of a new process. The new process (child process) is an exact copy of the calling process (parent process). This means the child process inherits the following attributes from the parent process:

- environment
- close-on-exec flag (see *exec*(S))
- signal handling settings (that is, **SIG\_DFL**, **SIG\_IGN**, **SIG\_HOLD**, function address)
- set-user-ID mode bit
- set-group-ID mode bit
- profiling on/off status
- nice value (see *nice*(S))
- all attached shared memory segments (see *shmop*(S))
- process group ID
- tty group ID (see *exit*(S))
- current working directory
- root directory
- file mode creation mask (see *umask*(S))
- file size limit (see *ulimit*(S))

The child process differs from the parent process in the following ways:

The child process has a unique process ID.

The child process has a different parent process ID (that is, the process ID of the parent process).

The child process has its own copy of the parent's file descriptors. Each of the child's file descriptors shares a common file pointer with the corresponding file descriptor of the parent.

All *semadj* values are cleared (see *semop*(S)).

Process locks, text locks, and data locks are not inherited by the child (see *plock*(S)).



The child process's *utime*, *stime*, *cutime*, and *cstime* are set to 0. The time left until an alarm clock signal is reset to 0.

The *fork* system call fails and no child process is created if one or more of the following is true:

- |          |                                                                                                               |
|----------|---------------------------------------------------------------------------------------------------------------|
| [EAGAIN] | The system-imposed limit on the total number of processes under execution would be exceeded.                  |
| [EAGAIN] | The system-imposed limit on the total number of processes under execution by a single user would be exceeded. |
| [EAGAIN] | Total amount of system memory available when reading via raw I/O is temporarily insufficient.                 |
| [EMFILE] | The shared data table would overflow.                                                                         |
| [ENOMEM] | The process requires more space than the system is able to supply.                                            |

## See Also

---

*exec(S)*, *nice(S)*, *plock(S)*, *ptrace(S)*, *semop(S)*, *shmop(S)*, *signal(S)*, *sigset(S)*, *times(S)*, *ulimit(S)*, *umask(S)*, *wait(S)*

## Diagnostics

---

Upon successful completion, *fork* returns a value of 0 to the child process and returns the process ID of the child process to the parent process. Otherwise, a value of -1 is returned to the parent process, no child process is created, and *errno* is set to indicate the error.

## Standards Conformance

---

*fork* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

# form

---

## FORM library routines

### Syntax

---

**#include <form.h>**

**cc [ flags ] files -lform -lcurses [ libraries ]**

**FORM \* new\_form (fields)**

**FIELD \*\* fields;**

**int free\_form (form)**

**FORM \* form;**

**int set\_new\_page (field, bool)**

**FIELD \* field;**

**int bool;**

**int new\_page (field)**

**FIELD \* field;**

**int set\_form\_fields (form, fields)**

**FORM \* form;**

**FIELD \*\* fields;**

**FIELD \*\* form\_fields (form)**

**FORM \* form;**

**int field\_count (form)**

**FORM \* form;**

**int set\_form\_win (form, window)**

**FORM \* form;**

**WINDOW \* window;**

**WINDOW \* form\_win (form)**

**FORM \* form;**

**int set\_form\_sub (form, window)**

**FORM \* form;**

**WINDOW \* window;**

**WINDOW \* form\_sub (form)**

**FORM \* form;**

**int set\_current\_field (form, field)**

**FORM \* form;**

**FIELD \* field;**

**FIELD \* current\_field (form)**  
**FORM \* form;**

**int field\_index(field)**  
**FIELD \* field;**

**int set\_form\_page (form, page)**  
**FORM \* form;**  
**int page;**

**int form\_page (form)**  
**FORM \* form;**

**int scale\_form (form, rows, cols)**  
**FORM \* form;**  
**int \* rows, cols;**

**typedef void (\* PTF\_void) ();**

**int set\_form\_init (form, func)**  
**FORM \* form;**  
**PTF\_void func;**

**PTF\_void form\_init (form)**  
**FORM \* form;**

**int set\_form\_term (form, func)**  
**FORM \* form;**  
**PTF\_void func;**

**PTF\_void form\_term (form)**  
**FORM \* form;**

**int set\_field\_init (form, func)**  
**FORM \* form;**  
**PTF\_void func;**

**PTF\_void field\_init (form)**  
**FORM \* form;**

**int set\_field\_term (form, func)**  
**FORM \* form;**  
**PTF\_void func;**

**PTF\_void field\_term (form)**  
**FORM \* form;**

**int post\_form (form)**  
**FORM \* form;**

**int unpost\_form (form)**  
**FORM \* form;**



```

int pos_form_cursor (form)
FORM * form;

int form_driver (form, c)
FORM * form;
int c;

int set_form_userptr (form, userptr)
FORM * form;
char * userptr;

char * form_userptr (form)
FORM * form;

int set_form_opts (form, opts)
FORM * form;
OPTIONS opts;

OPTIONS form_opts (form)
FORM * form;

int form_opts_on (form, opts)
FORM * form;
OPTIONS * opts;

int form_opts_off (form, opts)
FORM * form;
OPTIONS * opts;

```

## Description

---

FORM routines run on any terminal supported by *curses*(S), the low-level library. Once you compile your program **#include**ing the FORM header file `<form.h>`, you should link it with the **form** and **curses** library routines.

## Functions

---

The following is a list of FORM routines. For a complete description of each, see the *Programmer's Guide*.

**new\_form (fields)** creates a new form connected to the designated **fields** and returns a pointer to the form.

**free\_form (form)** disconnects the form from its associated field pointer array and deallocates the space for the form.

**set\_new\_page (field, bool)** marks the given field to begin a new page of the form.

**new\_page (field)** returns a boolean value indicating whether or not the given field begins a new page of the form.

**set\_form\_fields (form, fields)** changes the fields connected to **form** to **fields**.

**form\_fields (form)** returns a pointer to the field pointer array connected to **form**.

**field\_count (form)** returns the number of fields connected to **form**.

**set\_form\_win (form, window)** sets **window** as the form window of **form**.

**form\_win (form)** returns a pointer to the window associated with **form**.

**set\_form\_sub (form, window)** sets **window** as the form subwindow of **form**.

**form\_sub (form)** returns a pointer to the subwindow associated with **form**.

**set\_current\_field (form, field)** sets the current field of **form** to **field**.

**current\_field (form)** returns a pointer to the current field of **form**.

**field\_index(field)** returns the index in the field pointer array to the given **field**

**set\_form\_page (form, page)** sets the page number of **form** to **page**.

**form\_page (form)** returns the current page number of **form**.

**scale\_form (form, rows, cols)** returns the smallest window size necessary for **form**. **rows** and **cols** are pointers to the locations used to return the number of rows and columns for the form.

The workhorse of the forms subsystem, **form\_driver (form, c)**, checks if the character **c** is a form request or data. If it is a request, the form driver executes the request and reports the result. If it is data (a printable ASCII character), it enters the data into the current position in the current field. If it is not recognized, the form driver assumes it is an application-defined command and returns **E\_UNKNOWN\_COMMAND**.

The following **set** functions enable you to establish application routines to be executed automatically at initialization and termination points in your form application. You need not specify any application-defined initialization or termination routines at all, but they may be helpful for displaying messages or page numbers and other chores.



*freopen* is typically used to attach the preopened *streams* associated with *stdin*, *stdout*, and *stderr* to other files.

*fdopen* associates a *stream* with a file descriptor. File descriptors are obtained from *open*, *dup*, *creat*, or *pipe*(S), which open files but do not return pointers to a FILE structure *stream*. Streams are necessary input for many library routines. The *type* of *stream* must agree with the mode of the open file.

When a file is opened for update, both input and output may be done on the resulting *stream*. However, output may not be directly followed by input without an intervening *fseek* or *rewind*, and input may not be directly followed by output without an intervening *fseek*, *rewind*, or an input operation which encounters end-of-file.

When a file is opened for append (that is, when *type* is "a" or "a+"), it is impossible to overwrite information already in the file. The *fseek* function may be used to reposition the file pointer to any position in the file, but when output is written to the file, the current file pointer is disregarded. All output is written at the end of the file and causes the file pointer to be repositioned at the end of the output. If two separate processes open the same file for append, each process may write freely to the file without fear of destroying output being written by the other. The output from the two processes will be intermixed in the file in the order in which it is written.

## See Also

---

*creat*(S), *dup*(S), *open*(S), *pipe*(S), *fclose*(S), *fseek*(S), *stdio*(S)

## Diagnostics

---

*fopen*, *fdopen*, and *freopen* return a NULL pointer on failure.

## Standards Conformance

---

*fdopen* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

*fopen* and *freopen* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
ANSI X3.159-198X C Language Draft Standard, May 13, 1988;



IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;  
and NIST FIPS 151-1.

**set\_form\_init (form, func)** sets an application-defined initialization **func** to be called when the form is posted and just after a page change.

**form\_init (form)** returns a pointer to the initialization function, if any, called when the form is posted and just after a page change.

**set\_form\_term (form, func)** sets an application-defined **func** to be called when the form is unposted and just before a page change.

**form\_term (form)** returns a pointer to the termination function, if any, called when the form is unposted and just before a page change.

**set\_field\_init (form, func)** sets an application-defined **func** to be called when the form is posted and just after the current field changes.

**field\_init (form)** returns a pointer to the initialization function, if any, called when the form is posted and just after the current field changes.

**set\_field\_term (form, func)** sets **func** to be called when the form is unposted and just before the current field changes.

**field\_term (form)** returns a pointer to the termination function, if any, called when the form is unposted and just before the current field changes.

**post\_form (form)** writes the **form** in its associated subwindow.

**unpost\_form (form)** erases the **form** from its associated subwindow.

**pos\_form\_cursor (form)** moves the form window cursor to the location required by the form driver to resume form processing. This is sometimes helpful after you write a message or page number.

Every form has an associated user pointer that you can use to store pertinent data. **set\_form\_userptr (form, userptr)** sets the form's user pointer.

**form\_userptr (form)** returns the form's user pointer.

**set\_form\_opts (form, opts)** turns on the named options for the form and turns off all its remaining options. Options are boolean values. Currently, there are two form options, **O\_NL\_OVERLOAD** and **O\_BS\_OVERLOAD**.

**form\_opts (form)** returns the form's options setting.

**form\_opts\_on (form, opts)** turns on the named options.

**form\_opts\_off (form, opts)** turns off the named options.

## See Also

---

curses(S), field(S), fieldtype(S), item(S), panel(S), menu(S), tam(S)

## Diagnostics

---

The following values are returned by one or more routines that return an integer. For specific information on which routines return which value, see the *Programmer's Guide*.

|                          |                                              |
|--------------------------|----------------------------------------------|
| <b>E_OK</b>              | function returned successfully               |
| <b>E_CONNECTED</b>       | object is connected                          |
| <b>E_SYSTEM_ERROR</b>    | system error                                 |
| <b>E_BAD_ARGUMENT</b>    | argument is incorrect                        |
| <b>E_CURRENT</b>         | field is current field                       |
| <b>E_POSTED</b>          | form is posted                               |
| <b>E_INVALID_FIELD</b>   | field is invalid                             |
| <b>E_NOT_CONNECTED</b>   | object is not connected                      |
| <b>E_NO_ROOM</b>         | form does not fit in subwindow               |
| <b>E_BAD_STATE</b>       | called from inappropriate routine            |
| <b>E_UNKNOWN_COMMAND</b> | unknown command was given to the form driver |
| <b>E_REQUEST_DENIED</b>  | recognized request failed                    |



# **fpgetround, fpsetround, fpgetmask, fpsetmask, fpgetsticky, fpsetsticky**

IEEE floating point environment control

## **Syntax**

```
#include <ieeefp.h>
```

```
typedef enum {
```

```
 FP_RN=0, /* round to nearest */
 FP_RM, /* round to minus */
 FP_RP, /* round to plus */
 FP_RZ, /* round to zero (truncate) */
} fp_rnd;
```

```
fp_rnd fpgetround();
```

```
fp_rnd fpsetround(rnd_dir)
fp_rnd rnd_dir;
```

```
#define fp_except int
#define FP_X_INV 0x01 /* invalid operation exception */
#define FP_X_OFL 0x08 /* overflow exception */
#define FP_X_UFL 0x10 /* underflow exception */
#define FP_X_DZ 0x04 /* divide-by-zero exception */
#define FP_X_IMP 0x20 /* imprecise (loss of precision) */
#define FP_X_DNML 0x02 /* denormalization exception */
```

```
fp_except fpgetmask();
```

```
fp_except fpsetmask(mask);
fp_except mask;
```

```
fp_except fpgetsticky();
```

```
fp_except fpsetsticky(sticky);
fp_except sticky;
```

## **Description**

There are six floating point exceptions: divide-by-zero, overflow, underflow, imprecise (inexact) result, denormalization, and invalid operation. When a floating point exception occurs, the corresponding sticky bit is set (C), and if the mask bit is enabled (C), the trap takes place. These routines let the user change the behavior on occurrence of any of these exceptions, as well as change the rounding mode for floating point operations.

*fpgetround()* returns the current rounding mode.

*fpsetround()* sets the rounding mode and returns the previous rounding mode.

*fpgetmask()* returns the current exception masks.

*fpsetmask()* sets the exception masks and returns the previous setting.

*fpgetsticky()* returns the current exception sticky flags.

*fpsetsticky()* sets (clears) the exception sticky flags and returns the previous setting.

The default environment on the Intel 80386 processor family is:

Rounding mode set to nearest(FP\_RN),  
Divide-by-zero,  
Floating point overflow, and  
Invalid operation traps enabled.

## See Also

---

isnan(S)

## Warnings

---

*fpsetsticky()* modifies all sticky flags. *fpsetmask()* changes all mask bits.

C requires truncation (round to zero) for floating point to integral conversions. The current rounding mode has no effect on these conversions.

## Notes

---

One must clear the sticky bit to recover from the trap and to proceed. If the sticky bit is not cleared before the next floating point instruction is executed, a wrong exception type may be signaled.

For the same reason, when calling *fpsetmask()* the user should make sure that the sticky bit corresponding to the exception being enabled is cleared.

# fread, fwrite

binary input/output

## Syntax

```
#include <stdio.h>
#include <sys/types.h>
```

```
size_t fread (ptr, size, nitems, stream)
void *ptr;
int nitems;
size_t size;
FILE *stream;
```

```
size_t fwrite (ptr, size, nitems, stream)
void *ptr;
int nitems;
size_t size;
FILE *stream;
```

## Description

The *fread* function copies, into an array pointed to by *ptr*, *nitems* items of data from the named input *stream*, where an item of data is a sequence of bytes (not necessarily terminated by a null byte) of length *size*. *fread* stops appending bytes if an end-of-file or error condition is encountered while reading *stream*, or if *nitems* items have been read. *fread* leaves the file pointer in *stream*, if defined, pointing to the byte following the last byte read if there is one. *fread* does not change the contents of *stream*.

*fwrite* appends at most *nitems* items of data from the array pointed to by *ptr* to the named output *stream*. *fwrite* stops appending when it has appended *nitems* items of data or if an error condition is encountered on *stream*. *fwrite* does not change the contents of the array pointed to by *ptr*.

The argument *size* is typically *sizeof(\*ptr)* where the pseudo-function *sizeof* specifies the length of an item pointed to by *ptr*. If *ptr* points to a data type other than *char*, it should be cast into a pointer to *char*.

## See Also

read(S), write(S), fopen(S), getc(S), gets(S), printf(S), putc(S), puts(S), scanf(S), stdio(S)



## Diagnostics

---

The *fread* and *fwrite* functions return the number of items read or written. If *nitems* is non-positive, no characters are read or written, and 0 is returned by both *fread* and *fwrite*.

## Standards Conformance

---

*fread* and *fwrite* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

## frexp, ldexp, modf

---

manipulate parts of floating-point numbers

### Syntax

---

**double frexp (value, eptr)**  
double value;  
int \*eptr;

**double ldexp (value, exp)**  
double value;  
int exp;

**double modf (value, iptr)**  
double value, \*iptr;

### Description

---

Every non-zero number can be written uniquely as  $x * 2^n$ , where the “mantissa” (fraction)  $x$  is in the range  $0.5 \leq |x| < 1.0$ , and the “exponent”  $n$  is an integer. *frexp* returns the mantissa of a double *value* and stores the exponent indirectly in the location pointed to by *eptr*. If *value* is zero, both results returned by *frexp* are zero.

*ldexp* returns the quantity  $value * 2^{exp}$ .

*modf* returns the signed fractional part of *value* and stores the integral part indirectly in the location pointed to by *iptr*.

### Diagnostics

---

If *ldexp* would cause overflow, **±HUGE** (defined in <math.h>) is returned (according to the sign of *value*), and *errno* is set to **ERANGE**. If *ldexp* would cause underflow, zero is returned and *errno* is set to **ERANGE**.

### Standards Conformance

---

*frexp*, *ldexp* and *modf* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
ANSI X3.159-198X C Language Draft Standard, May 13, 1988;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

## **fseek, rewind, ftell**

---

reposition a file pointer in a stream

### **Syntax**

---

```
#include <stdio.h>
#include <unistd.h>

int fseek (stream, offset, ptrname)
FILE *stream;
long offset;
int ptrname;

void rewind (stream)
FILE *stream;

long ftell (stream)
FILE *stream;
```

### **Description**

---

The *fseek* function sets the position of the next input or output operation on the *stream*. The new position is at the signed distance *offset* bytes from the beginning, from the current position, or from the end of the file, according as *ptrname* has the value 0, 1, or 2, which is defined in the *<unistd.h>* header file as follows:

| <i>Name</i> | <i>Description</i>                                    |
|-------------|-------------------------------------------------------|
| SEEK_SET    | Set position equal to <i>offset</i> bytes.            |
| SEEK_CUR    | Set position to current location plus <i>offset</i> . |
| SEEK_END    | Set position to EOF plus <i>offset</i> .              |

*rewind(stream)* is equivalent to *fseek(stream, 0L, 0)*, except that no value is returned.

*fseek* and *rewind* undo any effects of *ungetc(S)*.

After *fseek* or *rewind*, the next operation on a file opened for update may be either input or output.

*ftell* returns the offset of the current byte relative to the beginning of the file associated with the named *stream*.

### **See Also**

---

*lseek(S)*, *fopen(S)*, *popen(S)*, *stdio(S)*, *ungetc(S)*



## Diagnostics

---

The *fseek* function returns non-zero for improper seeks, otherwise zero. An improper seek can be, for example, an *fseek* done on a file that has not been opened via *fopen*; in particular, *fseek* may not be used on a terminal or on a file opened via *popen*(S).

## Warning

---

Although on System V/386, an offset returned by *ftell* is measured in bytes, and it is permissible to seek to positions relative to that offset, portability to non-UNIX type systems requires that an offset be used by *fseek* directly. Arithmetic may not meaningfully be performed on such an offset, which is not necessarily measured in bytes.

## Standards Conformance

---

*fseek*, *ftell* and *rewind* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

# fsetpos

---

sets the file position indicator for a stream

## Syntax

---

```
#include <stdio.h>
```

```
int fsetpos(stream, pos)
```

```
FILE *stream;
```

```
const fpos_t *pos;
```

## Description

---

The *fsetpos* function sets the file position indicator for *stream* to the value of *pos*, which is obtained in a prior call to *fgetpos* against *stream*.

The function clears the end-of-file indicator and undoes any effects of the *ungetcstream*. After calling *fsetpos*, the next operation on *stream* may be either input or output.

## Return Value

---

If successful, the *fsetpos* function returns 0. On failure, the function returns a nonzero value and sets *errno* to one of the following manifest constants (defined in *stdio.h*):

| Constant | Meaning                                                                                            |
|----------|----------------------------------------------------------------------------------------------------|
| EINVAL   | An invalid <i>stream</i> value was passed.                                                         |
| EBADF    | The object that <i>stream</i> points to is not a valid file handle, or the file is not accessible. |

## See Also

---

fgetpos(S)

## Example

---

```
#include <stdio.h>
```

```
FILE *stream;
fpos_t pos[];
int val;
char list[100];
```

```
main()
```

```
{
 if ((stream = fopen("file1","rb")) == NULL) /* open file1 */
 printf ("Trouble opening file\n");
 else {
 fread(list,sizeof(char),100,stream); /* Read some data */
 if (fgetpos(stream,pos) != 0) /* Save current position */
 perror("fgetpos error");
 fread(list,sizeof(char),100,stream); /* Read some more */
 if (fsetpos(stream,pos) != 0) /* Return to saved position */
 perror("fsetpos error");
 }
}
```

This program opens the file named *file1* and reads 100 characters. It then uses *fgetpos* to find and save the file position pointer. After performing another read, the program calls *fsetpos* to restore the file pointer to the saved position.

## Standards Conformance

---

*fsetpos* is conformant with:

ANSI X3.159-198X C Language Draft Standard, May 13, 1988.



## ftw

---

walk a file tree

### Syntax

---

```
#include <ftw.h>
```

```
int ftw (path, fn, depth)
char *path;
int (*fn) ();
int depth;
```

### Description

---

The *ftw* function recursively descends the directory hierarchy rooted in *path*. For each object in the hierarchy, *ftw* calls *fn*, passing it a pointer to a null-terminated character string containing the name of the object, a pointer to a *stat* structure (see *stat(3)*) containing information about the object, and an integer. Possible values of the integer, defined in the *<ftw.h>* header file, are *FTW\_F* for a file, *FTW\_D* for a directory, *FTW\_DNR* for a directory that cannot be read, and *FTW\_NS* for an object for which *stat* could not successfully be executed. If the integer is *FTW\_DNR*, descendants of that directory will not be processed. If the integer is *FTW\_NS*, the *stat* structure will contain garbage. An example of an object that would cause *FTW\_NS* to be passed to *fn* would be a file in a directory with read but without execute (search) permission.

The *ftw* function visits a directory before visiting any of its descendants.

The tree traversal continues until the tree is exhausted, an invocation of *fn* returns a nonzero value, or some error is detected within *ftw* (such as an I/O error). If the tree is exhausted, *ftw* returns zero. If *fn* returns a nonzero value, *ftw* stops its tree traversal and returns whatever value was returned by *fn*. If *ftw* detects an error, it returns -1 and sets the error type in *errno*.

The *ftw* function uses one file descriptor for each level in the tree. The *depth* argument limits the number of file descriptors so used. If *depth* is zero or negative, the effect is the same as if it were 1. *depth* must not be greater than the number of file descriptors currently available for use. *ftw* will run more quickly if *depth* is at least as large as the number of levels in the tree.

## See Also

---

stat(S), malloc(S)

## Notes

---

Because *ftw* is recursive, it is possible for it to terminate with a memory fault when applied to very deep file structures.

The *ftw* function uses *malloc* to allocate dynamic storage during its operation. If *ftw* is forcibly terminated, such as by *longjmp* being executed by *fn* or an interrupt routine, *ftw* will not have a chance to free that storage, so it will remain permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have *fn* return a nonzero value at its next invocation.

## Standards Conformance

---

*ftw* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

# gamma

---

log gamma function

## Syntax

---

```
#include <math.h>

double gamma (x)
double x;

extern int signgam;
```

## Description

---

The *gamma* function returns  $\ln(|\Gamma(x)|)$ , where  $\Gamma(x)$  is defined as  $\int_0^{\infty} e^{-t} t^{x-1} dt$ . The sign of  $\Gamma(x)$  is returned in the external integer *signgam*. The argument  $x$  may not be a non-positive integer.

The following C program fragment might be used to calculate  $\Gamma$ :

```
if ((y = gamma(x)) > LN_MAXDOUBLE)
 error();
y = signgam * exp(y);
```

where LN\_MAXDOUBLE is the least value that causes *exp*(S) to return a range error, and is defined in the <values.h> header file.

## See Also

---

*exp*(S), *matherr*(S), *values*(M)

## Diagnostics

---

For non-negative integer arguments **HUGE** is returned, and *errno* is set to **EDOM**. A message indicating SING error is printed on the standard error output [for example, *gamma* (-5.0)].

If the correct value would overflow, *gamma* returns **HUGE** and sets *errno* to **ERANGE**.

These error-handling procedures may be changed with the function *matherr*(S).



## **Standards Conformance**

---

*gamma* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## **getc, getchar, fgetc, getw**

get character or word from a stream

---

### **Syntax**

---

```
#include <stdio.h>
```

```
int getc (stream)
FILE *stream;
```

```
int getchar ()
```

```
int fgetc (stream)
FILE *stream;
```

```
int getw (stream)
FILE *stream;
```

### **Description**

---

The *getc* function returns the next character (that is, byte) from the named input *stream*, as an integer. It also moves the file pointer, if defined, ahead one character in *stream*. *getchar* is defined as *getc(stdin)*. *getc* and *getchar* are macros.

The *fgetc* function behaves like *getc*, but is a function rather than a macro. *fgetc* runs more slowly than *getc*, but it takes less space per invocation and its name can be passed as an argument to a function.

The *getw* function returns the next word (that is, integer) from the named input *stream*. *getw* increments the associated file pointer, if defined, to point to the next word. The size of a word is the size of an integer and varies from machine to machine. *getw* assumes no special alignment in the file.

### **See Also**

---

*fclose(S)*, *ferror(S)*, *fopen(S)*, *fread(S)*, *gets(S)*, *putc(S)*, *scanf(S)*, *stdio(S)*

### **Diagnostics**

---

These functions return the constant **EOF** at end-of-file or upon an error. Because **EOF** is a valid integer, *ferror(S)* should be used to detect *getw* errors.

## Warning

---

If the integer value returned by *getc*, *getchar*, or *fgetc* is stored into a character variable and then compared against the integer constant EOF, the comparison may never succeed, because sign-extension of a character on widening to integer is machine-dependent.

## Notes

---

Because it is implemented as a macro, *getc* evaluates a *stream* argument more than once. In particular, *getc(\*f++)* does not work sensibly. *fgetc* should be used instead.

Because of possible differences in word length and byte ordering, files written using *putw* are machine-dependent, and may not be read using *getw* on a different processor.

## Standards Conformance

---

*fgetc*, *getc* and *getchar* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
ANSI X3.159-198X C Language Draft Standard, May 13, 1988;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

*getw* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



## getcwd

---

get path name of current working directory

### Syntax

---

```
char *getcwd (buf, size)
char *buf;
int size;
```

### Description

---

The *getcwd* function returns a pointer to the current directory path name. The value of *size* must be at least two greater than the length of the path name to be returned.

If *buf* is a NULL pointer, *getcwd* will obtain *size* bytes of space using *malloc*(S). In this case, the pointer returned by *getcwd* may be used as the argument in a subsequent call to *free*.

The function is implemented by using *popen*(S) to pipe the output of the *pwd*(C) command into the specified string space.

### Example

---

```
void exit(), perror();
.
.
.
if ((cwd = getcwd((char *)NULL, 64)) == NULL) {
 perror("pwd");
 exit(S);
}
printf("%s\n", cwd);
```

### See Also

---

*malloc*(S), *popen*(S), *pwd*(C)

### Diagnostics

---

Returns **NULL** with *errno* set if *size* is not large enough, or if an error occurs in a lower-level function.

[EINVAL]

If size is zero or not large enough to hold the path name.

## Standards Conformance

---

*getcwd* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

## getdents

---

read directory entries and put in a file-system-independent format

### Syntax

---

```
#include <sys/dirent.h>
```

```
int getdents (fildes, buf, nbyte)
```

```
int fildes;
```

```
char *buf;
```

```
unsigned nbyte;
```

### Description

---

The *fildes* argument is a file descriptor obtained from an *open*(S) or *dup*(S) system call.

The *getdents* system call attempts to read *nbyte* bytes from the directory associated with *fildes* and to format them as file system independent directory entries in the buffer pointed to by *buf*. Since the file-system-independent directory entries are of variable length, in most cases the actual number of bytes returned will be strictly less than *nbyte*.

The file-system-independent directory entry is specified by the *dirent* structure. For a description of this see *dirent* (F).

On devices capable of seeking, *getdents* starts at a position in the file given by the file pointer associated with *fildes*. Upon return from *getdents*, the file pointer is incremented to point to the next directory entry.

This system call was developed in order to implement the *readdir*(S) routine (for a description see *directory* (S)), and should not be used for other purposes.

The *getdents* system call will fail if one or more of the following is true:

- |          |                                                                |
|----------|----------------------------------------------------------------|
| [EBADF]  | <i>fildes</i> is not a valid file descriptor open for reading. |
| [EFAULT] | <i>buf</i> points outside the allocated address space.         |
| [EINVAL] | <i>nbyte</i> is not large enough for one directory entry.      |



|           |                                                                                            |
|-----------|--------------------------------------------------------------------------------------------|
| [ENOENT]  | The current file pointer for the directory is not located at a valid entry.                |
| [ENOLINK] | <i>fildev</i> points to a remote machine and the link to that machine is no longer active. |
| [ENOTDIR] | <i>fildev</i> is not a directory.                                                          |
| [EIO]     | An I/O error occurred while accessing the file system.                                     |

## See Also

---

directory(S), dirent(F)

## Diagnostics

---

Upon successful completion a non-negative integer is returned, indicating the number of bytes actually read. A value of 0 indicates the end of the directory has been reached. If the system call failed, a -1 is returned, and *errno* is set to indicate the error.

# getdvagent, getdvagnam, setdvagent, enddvagent, putdvag- nam, copydvagent

---

manipulate device assignment database entry

## Syntax

---

```
#include <sys/types.h>
#include <sys/security.h>
#include <sys/audit.h>
#include <prot.h>

struct dev_asg *getdvagent()

struct dev_asg *getdvagnam (name)
char *name;

void setdvagent()

void enddvagent()

int putdvagnam (name, dv)
char *name;
struct dev_asg *dv;

struct dev_asg *copydvagent(dv)
struct dev_asg *dv;
```

## Description

---

*getdvagent*, *getdvagnam*, and *copydvagent* each returns a pointer to an object with the following structure containing the broken-out fields of an entry in the Device Assignment database. Each database entry is returned as a *dev\_asg* structure, declared in the **<prot.h>** header file:

```
struct dev_field {
 char *fd_name; /* external name */
 char **fd_devs; /* device list */
 mask_t fd_type[1]; /* tape, printer, terminal */
 char *fd_max_sl; /* maximum sensitivity level */
 char *fd_min_sl; /* minimum sensitivity level */
 char *fd_cur_sl; /* currently assigned s.l. */
 mask_t fd_assign[1];
 /* single-level, multilevel, etc. */
};
```

```

/* bit offsets for fd_type */
#define AUTH_DEV_PRINTER0
 /* device is a printer (lp) */
#define AUTH_DEV_TERMINAL 1
 /* device is a terminal (login) */
#define AUTH_DEV_TAPE 2
 /* device can import/export data */

/* bit offsets for fd_assign */
#define AUTH_DEV_SINGLE 0
 /* single-level */
#define AUTH_DEV_MULTI 1
 /* multilevel */
#define AUTH_DEV_LABEL 2
 /* labeled import/export enabled */
#define AUTH_DEV_NOLABEL 3
 /* unlabeled import/export enabled */
#define AUTH_DEV_IMPORT 4
 /* enabled for import */
#define AUTH_DEV_EXPORT 5
 /* enabled for export */

/* this structure tells which of the corresponding fields
/* in dev_field are valid (filled).
*/
struct dev_flag {
 unsigned fg_name : 1,
 fg_devs : 1,
 fg_max_sl : 1,
 fg_min_sl : 1,
 fg_cur_sl : 1,
 fg_type : 1,
 fg_assign : 1,
 fg_reserved : 25;
};

struct dev_asg {
 struct dev_field ufld;
 struct dev_flag uflg;
};

```

The Device Assignment Database stores the relationship between device pathnames and real devices. Each entry contains a *name*, which is a cross reference to the terminal control database, and a list of *devices*, each of which is a pathname which corresponds to that device. Device drivers typically use different minor device numbers to correspond to different options on the same device, e.g., modem control on terminals or densities on tape drives. This list allows the device assignment software of the SMP to invalidate all references to a device when re-assigning it. The list is a table of character string pointers, whose last entry is a NULL pointer.



For SMP versions supporting mandatory access control, the database also stores the maximum, minimum, and current sensitivity levels of assigned devices, as well as the current assignment mode. On systems which do not support mandatory access control, the *fd\_min\_sl*, *fd\_max\_sl*, *fd\_cur\_sl*, and *fd\_assign* fields are not available. All sensitivity levels are stored as external representations (character strings). See the *mand(S)* page for routines that convert to internal representations.

*getdvagent* when first called returns a pointer to the first device assignment entry. Thereafter, it returns a pointer to the next entry, so successive calls can be used to search the database. *getdvagnam* searches from the beginning of the database until an entry with device name matching *name* is found, and returns a pointer to that entry. If an end of file or an error is encountered on reading, these functions return a NULL pointer. *copydvagent* copies a device assignment structure and the fields to which it refers to a newly-allocated data area. Since *getdvagent*, *getdvagnam*, and *putdvagent* re-use a static structure when accessing the database, the values of any entry must be saved if these routines are used again. The *dev\_asg* structure returned by *copydvagent* may be freed using *free* (see *malloc(S)*).

A call to *setdvagent* has the effect of setting the device assignment database back to the first entry, to allow repeated searches of the database. *Enddvagent* frees all memory and closes all files used to support these routines.

*putdvagnam* re-writes or adds an entry to the database. If there is an entry whose *fd\_name* field matches the *name* argument, that entry is replaced with the contents of the *dv* structure. Otherwise, that entry is added to the database.

## Files

---

/etc/auth/system/devassign

## Diagnostics

---

*getdvagent* and *getdvagnam* return a pointer to a static structure on success, or a NULL pointer on failure. This static structure is overwritten by *getdvagent*, *getdvagnam*, and *putdvagnam*. *putdvagnam* returns 1 on success, or 0 on failure. *copydvagent* returns a pointer to the newly-allocated structure on success, or a NULL pointer if there was a memory allocation error.

## Value Added

---

*getdvagent* is an extension of AT&T System V provided by the Santa Cruz Operation.

## getenv

---

return value for environment name

### Syntax

---

```
char *getenv (name)
char *name;
```

### Description

---

The *getenv* function searches the environment list [see *environ*(M)] for a string of the form *name* = *value* and returns a pointer to the *value* in the current environment if such a string is present, otherwise a NULL pointer.

### See Also

---

*exec*(S), *putenv*(S), *environ*(M).

### Standards Conformance

---

*getenv* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

# getgrent, getgrgid, getgrnam, setgrent, endgrent, fgetgrent

get group file entry

## Syntax

```
#include <grp.h>

struct group *getgrent ()

struct group *getgrgid (gid)
int gid;

struct group *getgrnam (name)
char *name;

void setgrent ()

void endgrent ()

struct group *fgetgrent (f)
FILE *f;
```

## Description

The *getgrent*, *getgrgid*, and *getgrnam* functions each return pointers to an object with the following structure containing the broken-out fields of a line in the */etc/group* file. Each line contains a “group” structure, defined in the *<grp.h>* header file.

```
struct group {
 char *gr_name; /* the name of the group */
 char *gr_passwd; /* the encrypted group password */
 int gr_gid; /* the numerical group ID */
 char **gr_mem; /* vector of pointers to member names */
};
```

The *getgrent* function when first called returns a pointer to the first group structure in the file; thereafter, it returns a pointer to the next group structure in the file; so, successive calls may be used to search the entire file. *getgrgid* searches from the beginning of the file until a numerical group ID matching *gid* is found and returns a pointer to the particular structure in which it was found. *getgrnam* searches from the beginning of the file until a group name matching *name* is found and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a NULL pointer.



A call to *setgrent* has the effect of rewinding the group file to allow repeated searches. *endgrent* may be called to close the group file when processing is complete.

*fgtgrent* returns a pointer to the next group structure in the stream *f*, which matches the format of */etc/group*.

## Files

---

*/etc/group*

## See Also

---

*getlogin(S)*, *getpwent(S)*, *group(F)*

## Diagnostics

---

A NULL pointer is returned on EOF or error.

## Warning

---

The above routines use *<stdio.h>*, which causes them to increase the size of programs, not otherwise using standard I/O, more than might be expected.

## Note

---

All information is contained in a static area, so it must be copied if it is to be saved.

## Standards Conformance

---

*endgrent*, *fgtgrent*, *getgrent*, and *setgrent* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

*getgrgid* and *getgrnam* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

# getgroups

---

get supplementary group ID's

## Syntax

---

```
#include <sys/types.h>

int getgroups(gidsetsize, gidset)
int gidsetsize;
gid_t gidset;
```

## Description

---

The *getgroups()* function fills in the array *grouplist* with the supplementary group IDs of the calling process. The *gidsetsize* argument specifies the number of elements in the supplied array *grouplist*. The actual number of supplementary group IDs is returned. The values of the array entries with indices larger than or equal to the returned value are undefined.

The effective group ID of the calling process is omitted from the returned list of supplementary group IDs.

As a special case, if the *gidsetsize* argument is zero, *getgroups()* returns the number of supplemental group IDs associated with the calling process without modifying the array pointed to by the *grouplist* argument.

## Return Value

---

Upon successful completion, the number of supplementary group IDs is returned. This value is zero if {NGROUPS\_MAX} is zero. A return value of -1 indicates failure and *errno* is set to indicate the error.

## Diagnostics

---

If any of the following conditions occur, the *getgroups()* function returns -1 and sets *errno* to the corresponding value:

- |          |                                                                                                             |
|----------|-------------------------------------------------------------------------------------------------------------|
| [EINVAL] | The <i>gidsetsize</i> argument is not equal to zero and is less than the number of supplementary group IDs. |
| [EFAULT] | The <i>grouplist</i> argument is a bad pointer into the calling process address space.                      |

## See Also

---

setgroups(S)

## Standards Conformance

---

*getgroups* is conformant with:

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;  
and NIST FIPS 151-1.



## gethz

---

return the frequency of the system clock in ticks per second

### Syntax

---

```
int gethz() ;
```

### Description

---

*gethz* returns the frequency of the system clock in ticks per second.

### See Also

---

*environ*(M)

### Warning

---

In the current implementation, *gethz* searches the environment list (see *environ*(M)) for a string of the form *HZ=val* , and returns *val*.

If *HZ* is not defined in the environment, or if *HZ* cannot be interpreted as a numeric integer, *gethz* returns 0.

### Note

---

The implementation of *gethz* may change.

## getlogin

---

get login name

### Syntax

---

```
char *getlogin ();
```

### Description

---

The *getlogin* function returns a pointer to the login name as found in */etc/utmp*. It may be used in conjunction with *getpwnam* to locate the correct password file entry when the same user ID is shared by several login names.

If *getlogin* is called within a process that is not attached to a terminal, it returns a **NULL** pointer. The correct procedure for determining the login name is to call *cuserid*, or to call *getlogin* and if it fails, to call *getpwuid*.

### Files

---

*/etc/utmp*

### See Also

---

*cuserid(S)*, *getgrent(S)*, *getpwent(S)*, *utmp(F)*

### Diagnostics

---

Returns the **NULL** pointer if *name* is not found.

### Note

---

The return values point to static data whose content is overwritten by each call.

## **Standards Conformance**

---

*getlogin* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;

and NIST FIPS 151-1.



## getluid

---

get login user ID

### Syntax

---

```
#include <sys/types.h>
#include <sys/security.h>
#include <sys/audit.h>
#include <prot.h>
```

```
int getluid ()
```

### Description

---

*Getluid* returns the login UID for the process.

The login UID, or LUID, is set at login or batch start time and once set cannot change. The LUID is an accurate representation of the user who logged into the system and cannot be altered during the session. The LUID is needed because both the effective and real UIDs can be altered by use of *setuid*(S) and the setuid bits on an executable file, and at times during a session, will not accurately reflect the login user.

*Getluid* will fail if the following is true:

[EPERM] The login UID has not yet been set for the process.

### RETURN VALUE

---

Upon successful completion the LUID is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

### See Also

---

*setluid*(S), *getuid*(S), *setuid*(S), *stat*(S)

### Value Added

---

*getluid* is an extension of AT&T System V provided by the Santa Cruz Operation.

## getmsg

---

get next message off a stream

### Syntax

---

```
#include <stropts.h>

int getmsg(fd, ctlptr, dataptr, flags)
int fd;
struct strbuf *ctlptr;
struct strbuf *dataptr;
int *flags;
```

### Description

---

The *getmsg* system call retrieves the contents of a message (see *intro(S)*) located at the *stream head* read queue from a STREAMS file, and places the contents into user-specified buffer(s). The message must contain either a data part, a control part, or both. The data and control parts of the message are placed into separate buffers, as described below. The semantics of each part is defined by the STREAMS module that generated the message.

The *fd* argument specifies a file descriptor referencing an open *stream*. *ctlptr* and *dataptr* each point to a *strbuf* structure which contains the following members:

```
int maxlen; /* maximum buffer length */
int len; /* length of data */
char *buf; /* ptr to buffer */
```

where *buf* points to a buffer in which the data or control information is to be placed, and *maxlen* indicates the maximum number of bytes this buffer can hold. On return, *len* contains the number of bytes of data or control information actually received, or is 0 if there is a zero-length control or data part, or is -1 if no data or control information is present in the message. *flags* may be set to the values 0 or RS\_HIPRI and is used as described below.

The *ctlptr* argument is used to hold the control part from the message and *dataptr* is used to hold the data part from the message. If *ctlptr* (or *dataptr*) is NULL or the *maxlen* field is -1, the control (or data) part of the message is not processed and is left on the *stream head* read queue, and *len* is set to -1. If the *maxlen* field is set to 0 and there is a zero-length control (or data) part, that zero-length part is removed from the read queue and *len* is set to 0. If the *maxlen* field is set to 0 and there are more than zero bytes of control (or data) information, that information is left on the read queue and *len* is set to 0. If the

*maxlen* field in *ctlptr* or *dataptr* is less than, respectively, the control or data part of the message, *maxlen* bytes are retrieved. In this case, the remainder of the message is left on the *stream head* read queue and a non-zero return value is provided, as described below under *Diagnostics*. If information is retrieved from a *priority* message, *flags* is set to RS\_HIPRI on return.

By default, *getmsg* processes the first priority or non-priority message available on the *stream head* read queue. However, a user may choose to retrieve only priority messages by setting *flags* to RS\_HIPRI. In this case, *getmsg* will only process the next message if it is a priority message.

If O\_NDELAY has not been set, *getmsg* blocks until a message, of the type(s) specified by *flags* (priority or either), is available on the *stream head* read queue. If O\_NDELAY has been set and a message of the specified type(s) is not present on the read queue, *getmsg* fails and sets *errno* to EAGAIN.

If a hangup occurs on the *stream* from which messages are to be retrieved, *getmsg* will continue to operate normally, as described above, until the *stream head* read queue is empty. Thereafter, it will return 0 in the *len* fields of *ctlptr* and *dataptr*.

The *getmsg* system call fails if one or more of the following is true:

- |           |                                                                                                                              |
|-----------|------------------------------------------------------------------------------------------------------------------------------|
| [EAGAIN]  | The O_NDELAY flag is set, and no messages are available.                                                                     |
| [EBADF]   | <i>fd</i> is not a valid file descriptor open for reading.                                                                   |
| [EBADMSG] | Queued message to be read is not valid for <i>getmsg</i> .                                                                   |
| [EFAULT]  | <i>ctlptr</i> , <i>dataptr</i> , or <i>flags</i> points to a location outside the allocated address space.                   |
| [EINTR]   | A signal was caught during the <i>getmsg</i> system call.                                                                    |
| [EINVAL]  | An illegal value was specified in <i>flags</i> , or the <i>stream</i> referenced by <i>fd</i> is linked under a multiplexer. |
| [ENOSTR]  | A <i>stream</i> is not associated with <i>fd</i> .                                                                           |

A *getmsg* can also fail if a STREAMS error message had been received at the *stream head* before the call to *getmsg*. The error returned is the value contained in the STREAMS error message.



## See Also

---

intro(S), read(S), poll(S), putmsg(S), write(S)

*The STREAMS Primer*

*The STREAMS Programmer's Guide*

## Diagnostics

---

Upon successful completion, a non-negative value is returned. A value of 0 indicates that a full message was read successfully. A return value of MORECTL indicates that more control information is waiting for retrieval. A return value of MOREDATA indicates that more data is waiting for retrieval. A return value of MORECTL|MOREDATA indicates that both types of information remain. Subsequent *getmsg* calls will retrieve the remainder of the message.

## Standards Conformance

---

*getmsg* is conformant with:

AT&T SVID Issue 2, Select Code 307-127.

## getopt

---

get option letter from argument vector

### Syntax

---

```
int getopt (argc, argv, optstring)
int argc;
char **argv, *optstring;
```

```
extern char *optarg;
extern int optind, opterr;
```

### Description

---

The *getopt* function returns the next option letter in *argv* that matches a letter in *optstring*. It supports all the rules of the command syntax standard (see *intro*(C)). So all new commands will adhere to the command syntax standard, they should use *getopts*(C) or *getopt*(S) to parse positional parameters and check for options that are legal for that command.

*optstring* must contain the option letters the command using *getopt* will recognize; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which must be separated from it by white space.

**optarg** is set to point to the start of the option-argument on return from *getopt*.

*getopt* places in **optind** the *argv* index of the next argument to be processed. **optind** is external and is initialized to 1 before the first call to *getopt*.

When all options have been processed (that is, up to the first non-option argument), *getopt* returns -1. The special option "--" may be used to delimit the end of the options; when it is encountered, -1 will be returned, and "--" will be skipped.

The following rules comprise the System V standard for command-line syntax:

**RULE 1**                      Command names must be between two and nine characters.

**RULE 2**                      Command names must include lowercase letters and digits only.

- RULE 3** Option names must be a single character in length.
- RULE 4** All options must be delimited by the - character.
- RULE 5** Options with no arguments may be grouped behind one delimiter.
- RULE 6** The first option-argument following an option must be preceded by white space.
- RULE 7** Option arguments cannot be optional.
- RULE 8** Groups of option arguments following an option must be separated by commas or separated by white space and quoted.
- RULE 9** All options must precede operands on the command line.
- RULE 10** The characters -- may be used to delimit the end of the options.
- RULE 11** The order of options relative to one another should not matter.
- RULE 12** The order of operands may matter and position-related interpretations should be determined on a command-specific basis.
- RULE 13** The - character preceded and followed by white space should be used only to mean standard input.

The function *getopt* is the command-line parser that will enforce the rules of this command syntax standard.



## Example

---

The following code fragment shows how one might process the arguments for a command that can take the mutually exclusive options **a** and **b**, and the option **o**, which requires an option-argument:

```
main (argc, argv)
int argc;
char **argv;
{
 int c;
 extern char *optarg;
 extern int optind;
 :
 while ((c = getopt(argc, argv, "abo:")) != -1)
 switch (c) {
 case 'a':
 if (bflg)
 errflg++;
 else
 aflg++;
 break;
 case 'b':
 if (aflg)
 errflg++;
 else
 bproc();
 break;
 case 'o':
 ofile = optarg;
 break;
 case '?':
 errflg++;
 }
 if (errflg) {
 (void) fprintf(stderr, "usage: . . . ");
 exit (1);
 }
 for (; optind < argc; optind++) {
 if (access(argv[optind], 4)) {
 :
 }
 }
}
```

## See Also

---

getopts(C), intro(C)

## Diagnostics

---

*getopt* prints an error message on standard error and returns a question mark (?) when it encounters an option letter not included in *optstring* or no option-argument after an option that expects one. This error message may be disabled by setting **opterr** to 0.

## Warning

---

Although the following command syntax rule (see *intro(C)*) relaxations are permitted under the current implementation, they should not be used because they may not be supported in future releases of the system. As in the *Example* section above, **a** and **b** are options, and the option **o** requires an option-argument:

cmd -abxxx file (Rule 5 violation: options with  
option-arguments must not be  
grouped with other options)

cmd -ab -oxxx file (Rule 6 violation: there must be  
white space after an option that  
takes an option-argument)

Changing the value of the variable **optind** or calling *getopt* with different values of *argv* may lead to unexpected results.

## Standards Conformance

---

*getopt* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

# getpass

---

read a password

## Syntax

---

```
char *getpass (prompt)
char *prompt;
```

## Description

---

The *getpass* function reads up to a new-line or EOF from the file */dev/tty* after prompting on the standard error output with the null-terminated string *prompt* and disabling echoing. A pointer is returned to a null-terminated string of at most 8 characters. If */dev/tty* cannot be opened, a NULL pointer is returned. An interrupt will terminate input and send an interrupt signal to the calling program before returning.

## Files

---

*/dev/tty*

## Warning

---

The above routine uses *<stdio.h>*, which causes it to increase the size of programs not otherwise using standard I/O more than might be expected.

## Note

---

The return value points to static data whose content is overwritten by each call.

## Standards Conformance

---

*getpass* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



# getpasswd

---

read or clear a password

## Syntax

---

```
#include <sys/types.h>
#include <sys/security.h>
#include <sys/audit.h>
#include <prot.h>

char *getpasswd (prompt, max_size)
char *prompt;
int max_size;

(void) getpasswd ((char *) 0, max_size)
char *prompt;
int max_size;
```

## Description

---

*Getpasswd*, when given a non-NULL and null-terminated *prompt*, reads up to a newline or EOF from the file */dev/tty*, after prompting on the standard error output with *prompt* and disabling echoing. A pointer is returned to a null-terminated string of at most *max\_size* characters. If */dev/tty* cannot be opened, a NULL pointer is returned. An interrupt will terminate input and send an interrupt signal to the calling program before returning.

The returned pointer above points to a static area. As soon as that password is no longer needed, *getpasswd* should again be invoked with a NULL prompt string. That will clear the password in the static area so that there is no chance of 'dirty memory' revealing passwords later on.

The *standard* *getpass(S)* routine may be implemented as:

```
char *
getpass (prompt)
char *prompt;
{
 return getpasswd(prompt, 8);
}
```

## Files

---

*/dev/tty*

## See Also

---

getpass(S)

## Notes

---

The return value points to static data whose content is overwritten by each call.

## Value Added

---

*getpasswd* is an extension of AT&T System V provided by the Santa Cruz Operation.

## **getpid, getpgrp, getppid**

---

get process, process group, and parent process IDs

### **Syntax**

---

**int** getpid ( )

**int** getpgrp ( )

**int** getppid ( )

### **Description**

---

The *getpid* system call returns the process ID of the calling process.

The *getpgrp* system call returns the process group ID of the calling process.

The *getppid* system call returns the parent process ID of the calling process.

### **See Also**

---

exec(S), fork(S), intro(S), setpgrp(S), signal(S)

### **Standards Conformance**

---

*getpgrp*, *getpid* and *getppid* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.



# getprcment, getprcmnam, setprcment, endprcment, putprcmnam

manipulate command control database entry

## Syntax

```
#include <sys/types.h>
#include <sys/security.h>
#include <sys/audit.h>
#include <prot.h>

struct pr_command *getprcment ()

struct pr_command *getprcmnam (name);
char *name;

void setprcment ();

void endprcment ();

int putprcmnam (name, pr)
char *name;
struct pr_command *pr;
```

## Description

*Getprcment* and *getprcmnam* each returns a pointer to an object with the following structure containing the broken-out fields of a line in the command control database. Only entries in the database dealing with users are scanned. Each line in the database contains a "pr\_command" structure, declared in the <prot.h> header file:

```
struct c_field {
 char fd_name[15]; /* holds command name */
 char fd_path[200]; /* xref to File Control Database */
 ushort fd_uid; /* uid of owner */
 ushort fd_gid; /* gid of group */
};

struct c_flag {
 unsigned
 fg_name:1, /* Is fd_name set? */
 fg_path:1, /* Is fd_path set? */
 fg_uid:1, /* Is fd_uid set? */
 fg_gid:1, /* Is fd_gid set? */
 fg_reserved:28; /* Reserved, leave set to 0 */
};
```

```

struct pr_command {
 struct c_field ufld;
 struct c_flag uflg;
 struct c_field sfld;
 struct c_flag sflg;
};

```

This structure is declared in `<prot.h>` so it is not necessary to redeclare it.

*Getprcment* when first called returns a pointer to the first user `pr_command` structure in the database; thereafter, it returns a pointer to the next `pr_command` structure in the database; so successive calls can be used to search the database. *Getprcmnam* searches from the beginning of the file until a login name matching name is found, and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a **NULL** pointer.

A call to *setprcment* has the effect of rewinding the command control file to allow repeated searches. *Endprcment* may be called to close the command control file when processing is complete.

*Putprcmnam* puts a new or replaced command control entry *pr* with key *name* into the appropriate file(s). If the "uflg.fg\_name" field is 0, the requested entry is deleted from the command control database. *Putprcmnam* locks the database for all update operations, and performs a *endprcment* after the update or failed attempt.

## Files

---

/etc/auth/commands

## See Also

---

getprpwent(S), getprtcnt(S), getprfient(S), getprdfent(S), authcap(S), authcap(F)

## Diagnostics

---

A **NULL** pointer is returned on **EOF** or error.

## Notes

---

All information is contained in a static area, so it must be copied if it is to be saved.

**Value Added**

---

*endprcment, getprcment, getprcmnam, putprcmnam* and *setprcment* are extensions of AT&T System V provided by the Santa Cruz Operation.



# getprdfent, getprdfnam, setprdfent, endprdfent, putprdfnam

---

manipulate default control database entry

## Syntax

---

```
#include <sys/types.h>
#include <sys/security.h>
#include <sys/audit.h>
#include <prot.h>

struct pr_default *getprdfent ()

struct pr_default *getprdfnam (name);
char *name;

void setprdfent ();

void endprdfent ();

int putprdfnam (name, pr)
char *name;
struct pr_default *pr;
```

## Description

---

*getprdfent* and *getprdfnam* each returns a pointer to an object with the following structure containing the broken-out fields of a line in the default control database. Only entries in the database dealing with users are scanned. Each line in the database contains a "pr\_default" structure, declared in the **<prot.h>** header file:

```
struct pr_default {
 char dd_name[20];
 char dg_name;
 struct pr_field prd;
 struct pr_flag prg;
 struct t_field tcd;
 struct t_flag tcg;
 struct f_field fid;
 struct f_flag fig;
 struct c_field cmd;
 struct c_flag cmg;
};
```

This structure is declared in **<prot.h>** so it is not necessary to redeclare it.

*getprdfent* when first called returns a pointer to the first user *pr\_default* structure in the database; thereafter, it returns a pointer to the next *pr\_default* structure in the database; so successive calls can be used to search the database. *getprdfnam* searches from the beginning of the file until a login name matching name is found, and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a NULL pointer.

A call to *setprdfent* has the effect of rewinding the default control file to allow repeated searches. *Endprdfent* may be called to close the default control file when processing is complete.

*Putprdfnam* puts a new or replaced default control entry *pr* with key *name* into the appropriate files(s). If the "uflg\_fg\_name" field is 0, the requested entry is deleted from the default control database. *Putprdfnam* locks the database for all update operations, and performs a *endprdfent* after the update or failed attempt.

## Files

---

/etc/auth/defaults

## See Also

---

getprpwent(S), getprtcent(S), getprfient(S), getprdfent(S), authcap(S), authcap(F)

## Diagnostics

---

A NULL pointer is returned on EOF or error.

## Notes

---

All information is contained in a static area, so it must be copied if it is to be saved.

## Value Added

---

*endprdfent*, *getprdfent*, *getprdfnam*, *putprdfnam* and *setprdfent* are extensions of AT&T System V provided by the Santa Cruz Operation.

# getprfient, getprfinam, setprfient, endprfient, putprfinam

manipulate file control database entry

## Syntax

```
#include <sys/types.h>
#include <sys/security.h>
#include <sys/audit.h>
#include <prot.h>

struct pr_file *getprfient ()

struct pr_file *getprfinam (name);
char *name;

void setprfient ();

void endprfient ();

int putprfinam (name, pr)
char *name;
struct pr_file *pr;
```

## Description

*getprfient* and *getprfinam* each returns a pointer to an object with the following structure containing the broken-out fields of a line in the file control database. Only entries in the database dealing with users are scanned. Each line in the database contains a “pr\_file” structure, declared in the <prot.h> header file:

```
struct f_field {
 char fd_name[200]; /* Holds full path name */
 ushort fd_uid; /* uid of owner */
 ushort fd_gid; /* gid of group */
 ushort fd_mode; /* permissions */
 char fd_type[2]; /* file type (one of r,b,c,d,f,s) */
 off_t fd_size; /* size in bytes */
 time_t fd_mtime; /* last time file noted as touched */
 time_t fd_ctime; /* last time inode noted as touched */
 long fd_cksum; /* checksum for file (see sum(C)) */
};
```



```

struct f_flag {
 unsigned
 fg_name:1, /* Is fd_name set? */
 fg_uid:1, /* Is fd_uid set? */
 fg_gid:1, /* Is fd_gid set? */
 fg_mode:1, /* Is fd_mode set? */
 fg_type:1, /* Is fd_type set? */
 fg_size:1, /* Is fd_size set? */
 fg_mtime:1, /* Is fd_mtime set? */
 fg_ctime:1, /* Is fd_ctime set? */
 fg_cksum:1, /* Is fd_cksum set? */
 fg_reserved:23; /* Reserved, leave set to 0 */
};

struct pr_file {
 struct f_field ufld;
 struct f_flag uflg;
 struct f_field sfld;
 struct f_flag sflg;
};

```

This structure is declared in `<prot.h>` so it is not necessary to redeclare it.

*getprfient* when first called returns a pointer to the first user *pr\_file* structure in the database; thereafter, it returns a pointer to the next *pr\_file* structure in the database; so successive calls can be used to search the database. *getprfinam* searches from the beginning of the file until a login name matching name is found, and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a **NULL** pointer.

A call to *setprfient* has the effect of rewinding the file control file to allow repeated searches. *Endprfient* may be called to close the file control file when processing is complete.

*Putprfinam* puts a new or replaced file control entry *pr* with key *name* into the appropriate file(s). If the "uflg.fg\_name" field is 0, the requested entry is deleted from the file control database. *Putprfinam* locks the database for all update operations, and performs a *endprfient* after the update or failed attempt.

## Files

---

/etc/auth/files

## See Also

---

getprpwent(S), getprtcent(S), getprcment(S), getprdfent(S),  
authcap(S), authcap(F)

## Diagnostics

---

A NULL pointer is returned on EOF or error.

## Notes

---

All information is contained in a static area, so it must be copied if it is to be saved.

## Value Added

---

*endprfient*, *getprfient*, *getprfinam*, *putprfinam* and *setprfient* are extensions of AT&T System V provided by the Santa Cruz Operation.

## getpriv

---

get system privileges associated with this process

### Syntax

---

```
#include <sys/types.h>
#include <sys/macros.h>
#include <sys/security.h>
#include <sys/audit.h>
```

```
int getpriv (privtype, privs)
int privtype;
priv_t *privs;
```

### Description

---

*getpriv* returns the system privilege vector for this process in the user-supplied *privs* vector. This vector should have at least **SEC\_SPRIVVEC\_SIZE** entries. The *privtype* argument may only contain the privilege type **SEC\_EFFECTIVE\_PRIV**.

The system privilege vector contains per-process privileges used by the TCB. See *setpriv*(S) for a description of the privileges currently defined.

*getpriv* will fail if the following is true:

[EFAULT]     *Privs* points to an invalid address.

[EINVAL]     *Privtype* is not **SEC\_EFFECTIVE\_PRIV**.

### RETURN VALUE

---

Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

### See Also

---

*setpriv*(S)

### Value Added

---

*getpriv* is an extension of AT&T System V provided by the Santa Cruz Operation.



# getprpwent, getprpwuid, getprpwnam, setprpwent, endprpwent, putprpwnam

---

manipulate protected password database entry

## Syntax

---

```
#include <sys/types.h>
#include <sys/security.h>
#include <sys/audit.h>
#include <prot.h>

struct pr_passwd *getprpwent ()

struct pr_passwd *getprpwuid (uid);
int uid;

struct pr_passwd *getprpwnam (name);
char *name;

void setprpwent ();

void endprpwent ();

int putprpwnam (name, pr)
char *name;
struct pr_passwd *pr;
```

## Description

---

*getprpwent*, *getprpwuid* and *getprpwnam* each returns a pointer to an object with the following structure containing the broken-out fields of a line in the protected password database. Only entries in the database dealing with users are scanned. Each line in the database contains a “pr\_passwd” structure, declared in the **<prot.h>** header file:

```

struct pr_field {
 /* Identity: */
 char fd_name[9];
 /* uses 8 character maximum from utmp */
 /* uid associated with name above */
 ushort fd_uid;
 char fd_encrypt[14];
 /* Encrypted password */
 /* user type - see user_type */
 priv_t fd_type;
 /* if a pseudo -user, the user behind it */
 char fd_owner[9];
 /* nice value with which to login */
 /* command privilege vector */
 int fd_nice
 mask_t fd_cprivs[AUTH_CPRIVEC_SIZE];
 /* system privilege vector */
 mask_t fd_sprivs[SEC_SPRIVEC_SIZE];
 /* audit control */
 mask_t fd_auditctl[AUTH_AUDITMASKVEC_SIZE];
 /* audit disposition */
 mask_t fd_auditdisp[AUTH_AUDITMASKVEC_SIZE];

 /* Password maintenance parameters: */
 /* min time between pswd changes */
 time_t fd_min;
 /* maximum length of password */
 int fd_maxlen;
 /* expiration time duration in secs */
 time_t fd_expire;
 /* account death time duration in secs */
 time_t fd_lifetime;
 /* last successful change */
 time_t fd_schange;
 /* last unsuccessful change */
 time_t fd_uchange;
 /* who can change this user's password */
 ushort fd_pswduser;
 /* can user pick his own passwords? */
 char fd_pick_pwd;
 /* can user get pswds generated for him? */
 char fd_gen_pwd;
 /* should generated pswds be restricted? */
 char fd_restrict;

 /* Login parameters: */
 /* last successful login */
 time_t fd_slogin;
 /* last unsuccessful login */
 time_t fd_ulin;
 /* consecutive unsuccessful logins */
 short fd_nlogins;
 /* maximum unsuc login tries allowed */
 short fd_max_tries;
 /* Unconditionally lock account? */
 char fd_lock;

```

```

/* System parameters: */
/* Require a password on single user shell? */
char fd_standpswd;
/* System security class */
mask_t fd_secclass[AUTH_SECCLASSEVEC_SIZE];
};

struct pr_flag {
 unsigned
/* Identity: */
 fg_name:1, /* Is fd_name set? */
 fg_uid:1, /* Is fd_uid set? */
 fg_encrypt:1, /* Is fd_encrypt set? */
 fg_type:1, /* Is fd_type set? */
 fg_owner:1, /* Is fd_owner set? */
 fg_nice:1, /* Is fd_nice set? */
 fg_cprivs:1, /* Is fd_sprivs set? */
 fg_sprivs:1, /* Is fd_sprivs set? */
 fg_auditcntl:1, /* Is fd_auditcntl set? */
 fg_auditdisp:1, /* Is fd_auditdisp set? */

/* Password maintenance parameters: */
 fg_min:1, /* Is fd_min set? */
 fg_maxlen:1, /* Is fd_maxlen set? */
 fg_expire:1, /* Is fd_expire set? */
 fg_lifetime:1, /* Is fd_lifetime set? */
 fg_schange:1, /* Is fd_schange set? */
 fg_uchange:1, /* Is fd_fchange set? */
 fg_pswduser:1, /* Is fd_pswduser set? */
 fg_pick_pwd:1, /* Is fd_pick_pwd set? */
 fg_gen_pwd:1, /* Is fd_gen_pwd set? */
 fg_restrict:1, /* Is fd_restrict set? */

/* Login parameters: */
 fg_slogin:1, /* Is fd_slogin set? */
 fg_ulin:1, /* Is fd_ulin set? */
 fg_nlogins:1, /* Is fd_nlogins set? */
 fg_max_tries:1, /* Is fd_max_tries set? */
 fg_lock:1, /* Is fd_lock set? */

 fg_standpswd:1, /* Is fd_standpswd set? */
 fg_secclass:1, /* Is fd_secclass set? */

 fg_reserved:5; /* Reserved, leave set to 0 */
};

struct pr_passwd {
 /* Fields assoc with this user */
 struct pr_field ufld;
 /* Flags assoc with this user */
 struct pr_flag uflg;
 /* Fields assoc with system */
 struct pr_field sfld;
 /* Flags assoc with system */
 struct pr_flag sflg;
};

```



This structure is declared in `<prot.h>` so it is not necessary to redeclare it.

*getprpwent* when first called returns a pointer to the first user `pr_passwd` structure in the database; thereafter, it returns a pointer to the next `pr_passwd` structure in the database; so successive calls can be used to search the database. Note that entries without a corresponding entry in `/etc/passwd` are skipped. The entries are scanned in the order they appear in `/etc/passwd`. *getprpwuid* searches from the beginning of the database until a numerical user id matching uid is found and returns a pointer to the particular structure in which it was found. *getprpwnam* searches from the beginning of the file until a login name matching name is found, and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a NULL pointer.

A call to *setprpwent* has the effect of rewinding the protected password files to allow repeated searches. *Endprpwent* may be called to close the protected password files when processing is complete.

*Putprpwnam* puts a new or replaced protected password entry *pr* with key *name* into the appropriate file(s). If the "uflag.fg\_name" field is 0, the requested entry is deleted from the protected password database. *Putprpwnam* locks the database for all update operations, and performs a *endprpwent* after the update or failed attempt.

## Files

---

`/etc/passwd`  
`/tcb/files/auth/*/*`

## See Also

---

`getpwent(S)`, `getprpwent(S)`, `getprtcent(S)`, `getprfient(S)`,  
`getprdfent(S)`, `authcap(S)`, `authcap(F)`

## Diagnostics

---

A NULL pointer is returned on EOF or error.

## Notes

---

All information is contained in a static area, so it must be copied if it is to be saved.

*getprpwent* assumed one name per UID and one UID per name. The sequential scan will loop between the first two instances of a multiple UID.

**Value Added**

---

*endprpwent, getprpwent, getprpwnam, getprpwuid, putprpwnam* and *setprpwent* are extensions of AT&T System V provided by the Santa Cruz Operation.

# getprtcnt, getprtcnam, setprtcnt, endprtcnt, putprtcnam

manipulate terminal control database entry

## Syntax

```
#include <sys/types.h>
#include <sys/security.h>
#include <sys/audit.h>
#include <prot.h>

struct pr_term *getprtcnt ()

struct pr_term *getprtcnam (name);
char *name;

void setprtcnt ();

void endprtcnt ();

int putprtcnam (name, pr)
char *name;
struct pr_term *pr;
```

## Description

*Getprtcnt* and *getprtcnam* each returns a pointer to an object with the following structure containing the broken-out fields of a line in the terminal control database. Only entries in the database dealing with users are scanned. Each line in the database contains a "pr\_term" structure, declared in the **<prot.h>** header file:

```
struct t_field {
 char fd_devname[12]; /* Same size as utmp entry */
 ushort fd_uid; /* uid of last successful login */
 time_t fd_slogin; /* time stamp of " " */
 ushort fd_uid; /* uid of last unsuccessful login */
 time_t fd_ulin; /* time stamp of " " */
 ushort fd_loutuid; /* uid of last logout */
 time_t fd_louttime; /* time stamp of " " */
 ushort fd_nlogins; /* consecutive failed attempts */
 ushort fd_max_tries; /* max unsuc login tries allowed */
 time_t fd_logdelay; /* delay between login tries */
 char fd_label[AUTH_TLSTSZ]; /* terminal label */
 char fd_lock; /* terminal locked? */
};
```



```

struct t_flag {
 unsigned
 fg_devname:1, /* Is fd_devname set? */
 fg_uid:1, /* Is fd_uid set? */
 fg_slogin:1, /* Is fd_stime set? */
 fg_uid:1, /* Is fd_uid set? */
 fg_ulogin:1, /* Is fd_ftime set? */
 fg_loutuid:1, /* Is fd_loutuid set? */
 fg_louttime:1, /* Is fd_louttime set? */
 fg_nlogins:1, /* Is fd_nlogins set? */
 fg_max_tries:1, /* Is fd_max_tries set? */
 fg_logdelay:1, /* Is fd_logdelay set? */
 fg_label:1, /* Is fd_label set? */
 fg_lock:1, /* Is fd_lock set? */
 fg_reserved:20; /* Reserved, leave set to 0 */
};

struct pr_term {
 struct t_field ufld;
 struct t_flag uflg;
 struct t_field sfld;
 struct t_flag sflg;
};

```

This structure is declared in **<prot.h>** so it is not necessary to redeclare it.

*Getprtcnt* when first called returns a pointer to the first user *pr\_term* structure in the database; thereafter, it returns a pointer to the next *pr\_term* structure in the database; so successive calls can be used to search the database. *Getprtcnam* searches from the beginning of the file until a login name matching name is found, and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a **NULL** pointer.

A call to *setprtcnt* has the effect of rewinding the terminal control file to allow repeated searches. *Endprtcnt* may be called to close the terminal control file when processing is complete.

*Putprtcnam* puts a new or replaced terminal control entry *pr* with key *name* into the appropriate file(s). If the "uflg.fg\_name" field is 0, the requested entry is deleted from the terminal control database. *Putprtcnam* locks the database for all update operations, and performs a *endprtcnt* after the update or failed attempt.

## Files

---

/etc/auth/ttyS

## See Also

---

getprpwent(S), getprfient(S), getprcment(S), getprdfent(S),  
authcap(S), authcap(F)

## DIAGNOSTICS

---

A NULL pointer is returned on EOF or error.

## Notes

---

All information is contained in a static area, so it must be copied if it is to be saved.

## Value Added

---

*endprtcent*, *getprtcent*, *getprtcnam*, *putprtcnam* and *setprtcent* are extensions of AT&T System V provided by the Santa Cruz Operation.

## getpw

---

get name from UID

### Syntax

---

```
int getpw (uid, buf)
int uid;
char *buf;
```

### Description

---

The *getpw* function searches the password file for a user ID number that equals *uid*, copies the line of the password file in which *uid* was found into the array pointed to by *buf*, and returns 0. *getpw* returns non-zero if *uid* cannot be found.

This routine is included only for compatibility with prior systems and should not be used; see *getpwent*(S) for routines to use instead.

### Files

---

/etc/passwd

### See Also

---

getpwent(S), passwd(F)

### Diagnostics

---

The *getpw* function returns non-zero on error.

### Warning

---

The above routine uses <stdio.h>, which causes it to increase, more than might be expected, the size of programs not otherwise using standard I/O.

### Standards Conformance

---

*getpw* is conformant with:

The X/Open Portability Guide II of January 1987.



# getpwent, getpwuid, getpwnam, setpwent, endpwent, fgetpwent

get password file entry

## Syntax

```
#include <pwd.h>

struct passwd *getpwent ()

struct passwd *getpwuid (uid)
int uid;

struct passwd *getpwnam (name)
char *name;

void setpwent ()

void endpwent ()

struct passwd *fgetpwent (f)
FILE *f;
```

## Description

The *getpwent*, *getpwuid*, and *getpwnam* functions each returns a pointer to an object with the following structure containing the broken-out fields of a line in the */etc/passwd* file. Each line in the file contains a “passwd” structure, declared in the *<pwd.h>* header file:

```
struct passwd {
 char *pw_name;
 char *pw_passwd;
 int pw_uid;
 int pw_gid;
 char *pw_age;
 char *pw_comment;
 char *pw_gecos;
 char *pw_dir;
 char *pw_shell;
};
```

This structure is declared in *<pwd.h>* so it is not necessary to redeclare it.

The fields have meanings described in *passwd(5)*.

The *getpwent* function when first called, returns a pointer to the first passwd structure in the file; thereafter, it returns a pointer to the next passwd structure in the file; so successive calls can be used to search the entire file. *getpwuid* searches from the beginning of the file until a numerical user ID matching *uid* is found and returns a pointer to the particular structure in which it was found. *getpwnam* searches from the beginning of the file until a login name matching *name* is found, and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a NULL pointer.

A call to *setpwent* has the effect of rewinding the password file to allow repeated searches. *endpwent* may be called to close the password file when processing is complete.

*fgetpwent* returns a pointer to the next passwd structure in the stream *f*, which matches the format of **/etc/passwd**.

---

## Files

/etc/passwd

---

## See Also

getlogin(S), getgrent(S), passwd(F)

---

## Diagnostics

A NULL pointer is returned on EOF or error.

---

## Warning

The above routines use <stdio.h>, which causes them to increase the size of programs, not otherwise using standard I/O, more than might be expected.

---

## Note

All information is contained in a static area, so it must be copied if it is to be saved.

---

## Standards Conformance

*endpwent*, *getpwent* and *setpwent* are conformant with:  
AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

*getpwnam* and *getpwuid* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;

and NIST FIPS 151-1.



## gets, fgets

---

get a string from a stream

### Syntax

---

```
#include <stdio.h>
```

```
char *gets (s)
char *s;
```

```
char *fgets (s, n, stream)
char *s;
int n;
FILE *stream;
```

### Description

---

The *gets* function reads characters from the standard input stream, *stdin*, into the array pointed to by *s*, until a new-line character is read or an end-of-file condition is encountered. The new-line character is discarded and the string is terminated with a null character.

The *fgets* function reads characters from the *stream* into the array pointed to by *s*, until *n*-1 characters are read, or a new-line character is read and transferred to *s*, or an end-of-file condition is encountered. The string is then terminated with a null character.

### See Also

---

ferror(S), fopen(S), fread(S), getc(S), scanf(S), stdio(S)

### Diagnostics

---

If end-of-file is encountered and no characters have been read, no characters are transferred to *s* and a NULL pointer is returned. If a read error occurs, such as trying to use these functions on a file that has not been opened for reading, a NULL pointer is returned. Otherwise *s* is returned.

## **Standards Conformance**

---

*fgets* and *gets* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

## **getuid, geteuid, getgid, getegid**

---

get real user, effective user, real group, and effective group IDs

### **Syntax**

---

**unsigned short getuid ( )**

**unsigned short geteuid ( )**

**unsigned short getgid ( )**

**unsigned short getegid ( )**

### **Description**

---

The *getuid* system call returns the real user ID of the calling process.

The *geteuid* system call returns the effective user ID of the calling process.

The *getgid* system call returns the real group ID of the calling process.

The *getegid* system call returns the effective group ID of the calling process.

### **See Also**

---

intro(S), setuid(S)

### **Standards Conformance**

---

*getegid*, *geteuid*, *getgid* and *getuid* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.



# getut: getutent, getutid, getutline, pututline, setutent, endutent, utmp-name

---

access utmp file entry

## Syntax

---

```
#include <utmp.h>

struct utmp *getutent ()

struct utmp *getutid (id)
struct utmp *id;

struct utmp *getutline (line)
struct utmp *line;

void pututline (utmp)
struct utmp *utmp;

void setutent ()

void endutent ()

void utmpname (file)
char *file;
```

## Description

---

The *getutent*, *getutid*, and *getutline* functions each return a pointer to a structure of the following type:

```
struct utmp {
 char ut_user[8]; /* User login name */
 char ut_id[4]; /* /etc/inittab id (usually line #) */
 char ut_line[12]; /* device name (console, lnxx) */
 short ut_pid; /* process id */
 short ut_type; /* type of entry */
 struct exit_status {
 short e_termination; /* Process termination status */
 short e_exit; /* Process exit status */
 } ut_exit; /* The exit status of a process
 * marked as DEAD_PROCESS. */
 time_t ut_time; /* time entry was made */
};
```

The *getutent* function reads in the next entry from a *utmp*-like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.

The *getutid* function searches forward from the current point in the *utmp* file until it finds an entry with a *ut\_type* matching *id->ut\_type* if the type specified is `RUN_LVL`, `BOOT_TIME`, `OLD_TIME` or `NEW_TIME`. If the type specified in *id* is `INIT_PROCESS`, `LOGIN_PROCESS`, `USER_PROCESS` or `DEAD_PROCESS`, then *getutid* will return a pointer to the first entry whose type is one of these four and whose *ut\_id* field matches *id->ut\_id*. If the end of file is reached without a match, it fails.

The *getutline* function searches forward from the current point in the *utmp* file until it finds an entry of the type `LOGIN_PROCESS` or `USER_PROCESS`, which also has a *ut\_line* string matching the *line->ut\_line* string. If the end of file is reached without a match, it fails.

*pututline* writes out the supplied *utmp* structure into the *utmp* file. It uses *getutid* to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of *pututline* will have searched for the proper entry using one of the *getut* routines. If so, *pututline* will not search. If *pututline* does not find a matching slot for the new entry, it will add a new entry to the end of the file.

*setutent* resets the input stream to the beginning of the file. This should be done before each search for a new entry if it is desired that the entire file be examined.

*endutent* closes the currently open file.

*utmpname* allows the user to change the name of the file examined, from `/etc/utmp` to any other file. It is most often expected that this other file will be `/etc/wtmp`. If the file does not exist, this will not be apparent until the first attempt to reference the file is made. *utmpname* does not open the file. It just closes the old file if it is currently open and saves the new file name.

---

## Files

`/etc/utmp`  
`/etc/wtmp`

---

## See Also

`ttyslot(S)`, `utmp(F)`

## Diagnostics

---

A NULL pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write.

## Notes

---

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. Each call to either *getutid* or *getutline* sees the routine examine the static structure before performing more I/O. If the contents of the static structure match what it is searching for, it looks no further. For this reason, to use *getutline* to search for multiple occurrences, it would be necessary to zero out the static after each success, or *getutline* would just return the same pointer over and over again.

There is one exception to the rule about removing the structure before further reads are done. The implicit read done by *pututline* (if it finds that it is not already at the correct place in the file) will not hurt the contents of the static structure returned by the *getutent*, *getutid*, or *getutline* routines, if the user has just modified those contents and passed the pointer back to *pututline*.

These routines use buffered standard I/O for input, but *pututline* uses an unbuffered non-standard write to avoid race conditions between processes trying to modify the *utmp* and *wtmp* files.

## Standards Conformance

---

*endutent*, *getutent*, *getutid*, *getutline*, *pututline*, *setutent* and *utmpname* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



# hsearch, hcreate, hdestroy

---

manage hash search tables

## Syntax

---

```
#include <search.h>
```

```
ENTRY *hsearch (item, action)
```

```
ENTRY item;
```

```
ACTION action;
```

```
int hcreate (nel)
```

```
int nel;
```

```
void hdestroy ()
```

## Description

---

The *hsearch* function is a hash-table search routine generalized from Knuth (6.4) Algorithm D. It returns a pointer into a hash table indicating the location at which an entry can be found. *item* is a structure of type ENTRY (defined in the < search.h > header file) containing two pointers: *item.key* points to the comparison key, and *item.data* points to any other data to be associated with that key. (Pointers to types other than character should be cast to pointer-to-character.) *action* is a member of an enumeration type ACTION indicating the disposition of the entry if it cannot be found in the table. **ENTER** indicates that the item should be inserted in the table at an appropriate point. **FIND** indicates that no entry should be made. Unsuccessful resolution is indicated by the return of a NULL pointer.

*hcreate* allocates sufficient space for the table and must be called before *hsearch* is used. *nel* is an estimate of the maximum number of entries that the table will contain. This number may be adjusted upward by the algorithm in order to obtain certain mathematically favorable circumstances.

*hdestroy* destroys the search table and may be followed by another call to *hcreate*.

## Example

---

The following example will read in strings followed by two numbers and store them in a hash table, discarding duplicates. It will then read in strings and find the matching entry in the hash table and print it out.

```

#include <stdio.h>
#include <search.h>

struct info { /* this is the info stored in the table */
 int age, room; /* other than the key. */
};
#define NUM_EMPL 5000 /* # of elements in search table */

main()
{
 /* space to store strings */
 char string_space[NUM_EMPL*20];
 /* space to store employee info */
 struct info info_space[NUM_EMPL];
 /* next avail space in string_space */
 char *str_ptr = string_space;
 /* next avail space in info_space */
 struct info *info_ptr = info_space;
 ENTRY item, *found_item, *hsearch();
 /* name to look for in table */
 char name_to_find[30];
 int i = 0;

 /* create table */
 (void) hcreate(NUM_EMPL);
 while (scanf("%s%d%d", str_ptr, &info_ptr->age,
 &info_ptr->room) != EOF && i++ < NUM_EMPL) {
 /* put info in structure, and structure in item */
 item.key = str_ptr;
 item.data = (char *)info_ptr;
 str_ptr += strlen(str_ptr) + 1;
 info_ptr++;
 /* put item into table */
 (void) hsearch(item, ENTER);
 }
 /* access table */
 item.key = name_to_find;
 while (scanf("%s", item.key) != EOF) {
 if ((found_item = hsearch(item, FIND)) != NULL) {
 /* if item is in the table */
 (void)printf("found %s, age = %d, room = %d\n",
 found_item->key,
 ((struct info *)found_item->data)->age,
 ((struct info *)found_item->data)->room);
 } else {
 (void)printf("no such employee %s\n",
 name_to_find);
 }
 }
}

```

## See Also

---

bsearch(S), lsearch(S), malloc(S), string(S), tsearch(S)

## Diagnostics

---

The *hsearch* function returns a NULL pointer if either the action is **FIND** and the item could not be found, or the action is **ENTER** and the table is full.

*hcreate* returns zero if it cannot allocate sufficient space for the table.

## Notes

---

The *hsearch* function uses *open addressing* with a *multiplicative* hash function. However, its source code has many other options available which the user may select by compiling the *hsearch* source with the following symbols defined to the preprocessor:

**DIV** Use the *remainder modulo table size* as the hash function instead of the multiplicative algorithm.

**USCR** Use a User-Supplied Comparison Routine for ascertaining table membership. The routine should be named *hcompare* and should behave in a manner similar to *strcmp* (see *string(S)*).

**CHAINED** Use a linked list to resolve collisions. If this option is selected, the following other options become available.

**START** Place new entries at the beginning of the linked list (default is at the end).

**SORTUP** Keep the linked list sorted by key in ascending order.

**SORTDOWN** Keep the linked list sorted by key in descending order.

Additionally, there are preprocessor flags for obtaining debugging printout (**-DDEBUG**) and for including a test driver in the calling routine (**-DDRIVER**). The source code should be consulted for further details.

## Warning

---

*hsearch* and *hcreate* use *malloc(S)* to allocate space.

## Note

---

Only one hash search table may be active at any given time.



## **Standards Conformance**

---

*hcreate*, *hdestroy* and *hsearch* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

# hypot

---

euclidean distance function

## Syntax

---

```
#include <math.h>
```

```
double hypot (x, y)
double x, y;
```

## Description

---

*hypot* returns

$\text{sqrt}(x * x + y * y)$ ,

taking precautions against unwarranted overflows.

## See Also

---

*matherr*(S)

## Diagnostics

---

When the correct value would overflow, *hypot* returns **HUGE** and sets *errno* to **ERANGE**.

These error-handling procedures may be changed with the function *matherr*(S).

## Standards Conformance

---

*hypot* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## identity

---

get or check uids or gids from program start

### Syntax

---

```
#include <sys/types.h>
#include <sys/security.h>
#include <sys/audit.h>
#include <prot.h>
```

```
ushort starting_luid ()
```

```
ushort starting_ruid ()
```

```
ushort starting_euid ()
```

```
ushort starting_rgid ()
```

```
ushort starting_egid ()
```

```
int is_starting_luid (uid)
ushort uid;
```

```
int is_starting_ruid (uid)
ushort uid;
```

```
int is_starting_euid (uid)
ushort uid;
```

```
int is_starting_rgid (gid)
ushort gid;
```

```
int is_starting_egid (gid)
ushort gid;
```

```
void set_auth_parameters (argc, argv)
int argc;
char *argv[];
```

```
void check_auth_parameters ()
```

### Description

---

These routines provide a way to recall the IDs of a process at the time the program started. They are useful when interrogating the invoking environment of a program after any *setuid*(S) or *setgid*(S) calls have been made so that the original environment can be captured.



*Starting\_luid* returns the login UID for the process as it was set in the beginning of the program. The login UID is the immutable stamp for the process and accurately denotes the account under which the session is being run, regardless of subsequent *setuid*(S) calls. *Starting\_ruid* returns the real UID for the process as it was set in the beginning of the program. Similarly, *starting\_euid* returns the effective UID, *starting\_rgid* returns the real GID, and *starting\_egid* returns the effective GID. These IDs may not be the same as those returned by *getluid*(S), *getuid*(S), *geteuid*(S), *getgid*(S), or *getegid*(S), respectively, because intervening calls to *setluid*(S), *setuid*(S) or *setgid*(S) can change the latter set while the former set remains the same.

The routine *is\_starting\_luid* returns 1 if the argument is the same as the login UID at the time when *set\_auth\_parameters* was invoked, and 0 otherwise. Similarly, *is\_starting\_ruid* returns 1 if the argument is the same as the real UID at the time when *set\_auth\_parameters* was invoked, and 0 otherwise; *is\_starting\_euid* returns 1 if the argument is the same as the effective UID at the time when *set\_auth\_parameters* was invoked, and 0 otherwise; *is\_starting\_rgid* returns 1 if the argument is the same as the real GID at the time when *set\_auth\_parameters* was invoked, and 0 otherwise; and *is\_starting\_egid* returns 1 if the argument is the same as the effective GID at the time when *set\_auth\_parameters* was invoked, and 0 otherwise.

The *set\_auth\_parameters* routine is used to retain the IDs for future lookup. It also tests the kernel to see if the C2 security features have been loaded. If not, the program exists with an error message. It should be called first in a program or there is a chance that it will capture an environment different from the program beginning. The two arguments are the argument count and vector with which the program was called. *Check\_auth\_parameters* will verify that *set\_auth\_parameters* has been previously invoked. If not, the program exists. If so, nothing happens.

## Notes

---

These routines only work as advertised when *set\_auth\_parameters* is called as the first item in *main*().

## See Also

---

*getuid*(S), *geteuid*(S), *getgid*(S), *getegid*(S)

## Value Added

---

*identity* is an extension of AT&T System V provided by the Santa Cruz Operation.

## ioctl

control device

### Syntax

```
int ioctl (fildes, request, arg)
int fildes, request;
```

### Description

The *ioctl* system call performs a variety of control functions on devices and STREAMS. For non-STREAMS files, the functions performed by this call are *device-specific* control functions. The arguments *request* and *arg* are passed to the file designated by *fildes* and are interpreted by the device driver. This control is infrequently used on non-STREAMS devices, with the basic input/output functions performed through the *read*(S) and *write*(S) system calls.

*fildes* is an open file descriptor that refers to a device. *request* selects the control function to be performed and will depend on the device being addressed. *arg* represents additional information that is needed by this specific device to perform the requested function. The data type of *arg* depends upon the particular control request, but it is either an integer or a pointer to a device-specific data structure.

In addition to device-specific and STREAMS functions, generic functions are provided by more than one device driver, for example, the general terminal interface (see *termio*(M)).

The *ioctl* system call will fail for any type of file if one or more of the following is true:

- [EBADF]            *fildes* is not a valid open file descriptor.
- [ENOTTY]          *fildes* is not associated with a device driver that accepts control functions.
- [EINTR]           A signal was caught during the *ioctl* system call.

The *ioctl* system call will also fail if the device driver detects an error. In this case, the error is passed through *ioctl* without change to the caller. A particular driver might not have all of the following error cases. Other requests to device drivers will fail if one or more of the following is true:

- [EFAULT]          *Request* requires a data transfer to or from a buffer pointed to by *arg*, but some part of the buffer is outside the process's allocated space.



|           |                                                                                                                                                 |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| [EINVAL]  | <i>Request</i> or <i>arg</i> is not valid for this device.                                                                                      |
| [EIO]     | Some physical I/O error has occurred.                                                                                                           |
| [ENXIO]   | The <i>request</i> and <i>arg</i> are valid for this device driver, but the service requested cannot be performed on this particular subdevice. |
| [ENOLINK] | <i>Fildes</i> is on a remote machine and the link to that machine is no longer active.                                                          |

STREAMS errors are described in *streamio(7)*.

## See Also

---

*termio(M)*

## Diagnostics

---

Upon successful completion, the value returned depends upon the device control function, but must be a non-negative integer. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## Standards Conformance

---

*ioctl* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



## **isnan: isnand, isnanf**

---

test for floating point NaN (Not-A-Number)

### **Syntax**

---

```
#include <ieeefp.h>
```

```
int isnand (dsrc)
double dsrc;
```

```
int isnanf (fsrc)
float fsrc;
```

### **Description**

---

The *isnand* and *isnanf* functions return true (C) if the argument *dsrc* or *fsrc* is a NaN; otherwise they return false (0).

Neither routine generates any exception, even for signaling NaNs.

*isnanf()* is implemented as a macro included in <ieeefp.h>.

### **See Also**

---

fpgetround(S)

## item

---

### CRT item routines

### Syntax

---

```
#include <menu.h>
```

```
cc [flags] files -lmenu -lcurses [libraries]
```

```
ITEM *new_item(n, d)
```

```
ITEM *item;
```

```
char *n, *d;
```

```
int free_item(i)
```

```
ITEM *i;
```

```
char *item_name(i)
```

```
ITEM *i;
```

```
char *item_description(i)
```

```
ITEM *i;
```

```
int set_item_opts(i, o)
```

```
ITEM *item;
```

```
OPTIONS o;
```

```
OPTIONS item_opts(i)
```

```
ITEM *i;
```

```
int item_opts_on (item, opts)
```

```
ITEM *item;
```

```
OPTIONS opts;
```

```
int item_opts_off (item, opts)
```

```
ITEM *item;
```

```
OPTIONS opts;
```

```
int set_item_value(i, c)
```

```
ITEM *item;
```

```
int *c;
```

```
int item_value(i)
```

```
ITEM *item;
```

```
int set_item_userptr(i, n)
```

```
ITEM *item;
```

```
char *n;
```

```
char *item_userptr(i)
```

```
ITEM *i;
```

```
int item_count(m)
MENU *m;
```

```
int item_visible(i)
ITEM *item;
```

## Description

---

These routines allow you to create, display, and access items. Menus can be displayed on any display device supported by the low-level () library *curses*(S). Once you compile your program including the ITEM header file **menu.h**, you should link it with the ITEM and *curses* library routines.

## Functions

---

*new\_item* ( *n* , *d* ) creates a new item with name *n* and description *d*. It returns a pointer to the new item. In general, you should store these item field pointers in an array.

*free\_item*() frees the storage allocated for the given item. Once an item is freed, you can no longer connect it to a menu.

*item\_name* ( *i* ) returns a pointer to the given item's name.

*item\_description* ( *i* ) returns a pointer to the given item's description.

*set\_item\_opts* ( *i* , *o* ) turns on the named option(s) for the item and turns off its remaining options, if any. Options are boolean values. Currently, there is one item option O\_SELECTABLE which enables your end-user to select the item. The initial current default is to have O\_SELECTABLE on for every item.

*item\_opts* ( *i* ) returns the given item's option(s) setting. To set options, you can apply boolean operators to the value returned by *item\_opts*() and let the result be the second argument to *set\_item\_opts*().

*item\_opts\_on* ( *item* , *opts* ) turns on the named options for the item.

*item\_opts\_off* ( *item* , *opts* ) turns off the named options for the item.

Unlike single-valued menus, multi-valued menus enable your end-user to select one or more items from a menu. *set\_item\_value* ( *i* , *c* ) sets the given item's select value—TRUE (selected) or FALSE (not selected). To make a menu multi-valued, you use *set\_menu\_opts*() or *menu\_opts\_off*() to turn off option O\_ONEVALUE. *set\_item\_value*() may be used only with multi-valued menus.



*item\_value ( i )* returns the select value of the given item, either TRUE (selected) or FALSE (unselected).

Every item has an associated user pointer that you can use to store pertinent information. *set\_item\_userptr ( i , n )* sets the item's user pointer.

*item\_userptr ( i )* returns the item's user pointer.

*item\_count ( m )* returns the number of items in the given menu.

A menu item is visible if it currently appears in the subwindow of the posted menu to which it is connected. If an item is visible, *item\_visible ( i )* returns TRUE. If not, it returns FALSE.

## See Also

---

curses(S), field(S), fieldtype(S), form(S), menu(S), panel(S), tam(S)

## Diagnostics

---

The following values are returned by one or more routines that return an integer.

|                          |                                                 |
|--------------------------|-------------------------------------------------|
| <b>E_OK</b>              | routine returned normally                       |
| <b>E_SYSTEM_ERROR</b>    | system error                                    |
| <b>E_BAD_ARGUMENT</b>    | an incorrect argument was passed to the routine |
| <b>E_POSTED</b>          | menu is already posted                          |
| <b>E_CONNECTED</b>       | one or more items are connected to another menu |
| <b>E_BAD_STATE</b>       | routine called from an inappropriate routine    |
| <b>E_NO_ROOM</b>         | menu does not fit within its subwindow          |
| <b>E_NOT_POSTED</b>      | menu has not yet been posted                    |
| <b>E_UNKNOWN_COMMAND</b> | unrecognizable request was given to the driver  |
| <b>E_NO_MATCH</b>        | no match occurred                               |

ITEM (S)

**E\_NOT\_SELECTABLE**

item cannot be selected

**E\_NOT\_CONNECTED**

no items are associated with the menu

**E\_REQUEST\_DENIED**

menu driver could not process the request

ITEM (S)

## kill

---

send a signal to a process or a group of processes

### Syntax

---

```
#include <signal.h>
```

```
int kill (pid, sig)
int pid, sig;
```

### Description

---

The *kill* system call sends a signal to a process or a group of processes. The process or group of processes to which the signal is to be sent is specified by *pid*. The signal that is to be sent is specified by *sig* and is either one from the list given in *signal(S)* or 0. If *sig* is 0 (the null signal), error checking is performed but no signal is actually sent. This can be used to check the validity of *pid*.

The real or effective user ID of the sending process must match the real or effective user ID of the receiving process, unless the effective user ID of the sending process is super-user.

The processes with a process ID of 0 and a process ID of 1 are special processes (see *intro(S)*) and will be referred to below as *proc0* and *proc1*, respectively.

If *pid* is greater than zero, *sig* will be sent to the process whose process ID is equal to *pid*. *pid* may equal 1.

If *pid* is 0, *sig* will be sent to all processes excluding *proc0* and *proc1* whose process group ID is equal to the process group ID of the sender.

If *pid* is -1 and the effective user ID of the sender is not super-user, *sig* will be sent to all processes excluding *proc0* and *proc1* whose real user ID is equal to the effective user ID of the sender.

If *pid* is -1 and the effective user ID of the sender is super-user, *sig* will be sent to all processes excluding *proc0* and *proc1*.

If *pid* is negative but not -1, *sig* will be sent to all processes whose process group ID is equal to the absolute value of *pid*.



The *kill* system call will fail and no signal will be sent if one or more of the following is true:

- |          |                                                                                                                                                                |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [EINVAL] | <i>sig</i> is not a valid signal number.                                                                                                                       |
| [EINVAL] | <i>sig</i> is SIGKILL and <i>pid</i> is 1 (proc1).                                                                                                             |
| [ESRCH]  | No process can be found corresponding to that specified by <i>pid</i> .                                                                                        |
| [EPERM]  | The user ID of the sending process is not super-user, and its real or effective user ID does not match the real or effective user ID of the receiving process. |

## See Also

---

getpid(S), setpggrp(S), signal(S), sigset(S), kill(C)

## Diagnostics

---

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## Standards Conformance

---

*kill* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

## **l3tol, ltol3**

---

convert between 3-byte integers and long integers

### **Syntax**

---

```
void l3tol (lp, cp, n)
long *lp;
char *cp;
int n;
```

```
void ltol3 (cp, lp, n)
char *cp;
long *lp;
int n;
```

### **Description**

---

The *l3tol* function converts a list of *n* three-byte integers packed into a character string pointed to by *cp* into a list of long integers pointed to by *lp*.

*ltol3* performs the reverse conversion from long integers (*lp*) to three-byte integers (*cp*).

These functions are useful for file-system maintenance where the block numbers are three bytes long.

### **See Also**

---

fs(F).

### **Note**

---

Because of possible differences in byte ordering, the numerical values of the long integers are machine-dependent.

### **Standards Conformance**

---

*l3tol* and *ltol3* are conformant with:

The X/Open Portability Guide II of January 1987.

## labs

---

converts to absolute value

### Syntax

---

```
#include <stdlib.h>
```

```
long labs(n)
long n;
```

### Description

---

The *labs* function produces the absolute value of its long-integer argument *n*.

### Return Value

---

The *labs* function returns the absolute value of its argument. There is no error return.

### See Also

---

*abs*(S), *fabs*(S)

### Example

---

```
#include <stdlib.h>
#include <stdio.h>

main()
{
 long x,y;

 x = -41567L;
 y = labs(x);
 printf("The labs(%ld) = %ld \n",x,y);
}
```

This program uses *labs* to get and display the absolute value of -41,567.

### Standards Conformance

---

*labs* is conformant with:

ANSI X3.159-198X C Language Draft Standard, May 13, 1988.



## ldahread

---

read the archive header of a member of an archive file

### Syntax

---

```
#include <stdio.h>
#include <ar.h>
#include <filehdr.h>
#include <ldfcn.h>
```

```
int ldahread (ldptr, arhead)
LDFILE *ldptr;
ARCHDR *arhead;
```

### Description

---

If **TYPE**(*ldptr*) is the archive file magic number, *ldahread* reads the archive header of the common object file currently associated with *ldptr* into the area of memory beginning at *arhead*.

*ldahread* returns **SUCCESS** or **FAILURE**. *ldahread* will fail if **TYPE**(*ldptr*) does not represent an archive file, or if it cannot read the archive header.

The program must be loaded with the object file access routine library **libld.a**.

### See Also

---

ldclose(S), ldopen(S), ldfcn(F), ar(F).

## **ldclose, ldaclose**

---

close a common object file

### **Syntax**

---

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>
```

```
int ldclose (ldptr)
LDFILE *ldptr;
```

```
int ldaclose (ldptr)
LDFILE *ldptr;
```

### **Description**

---

The *ldopen*(S) and *ldclose* functions are designed to provide uniform access to both simple object files and object files that are members of archive files. Thus an archive of common object files can be processed as if it were a series of simple common object files.

If **TYPE**(*ldptr*) does not represent an archive file, *ldclose* will close the file and free the memory allocated to the **LDFILE** structure associated with *ldptr*. If **TYPE**(*ldptr*) is the magic number of an archive file, and if there are any more files in the archive, *ldclose* will reinitialize **OFFSET**(*ldptr*) to the file address of the next archive member and return **FAILURE**. The **LDFILE** structure is prepared for a subsequent *ldopen*(S). In all other cases, *ldclose* returns **SUCCESS**.

*ldaclose* closes the file and frees the memory allocated to the **LDFILE** structure associated with *ldptr* regardless of the value of **TYPE**(*ldptr*). *ldaclose* always returns **SUCCESS**. The function is often used in conjunction with *ldaopen*.

The program must be loaded with the object file access routine library **libld.a**.

### **See Also**

---

*fclose*(S), *ldopen*(S), *ldfcn*(F)

## Ldfhread

---

read the file header of a common object file

### Syntax

---

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>
```

```
int ldfhread (ldptr, filehead)
LDFILE *ldptr;
FILHDR *filehead;
```

### Description

---

The *ldfhread* function reads the file header of the common object file currently associated with *ldptr* into the area of memory beginning at *filehead*.

*ldfhread* returns **SUCCESS** or **FAILURE**. *ldfhread* will fail if it cannot read the file header.

In most cases the use of *ldfhread* can be avoided by using the macro **HEADER(*ldptr*)** defined in **ldfcn.h** (see *ldfcn* (F)). The information in any field, *fieldname*, of the file header may be accessed using **HEADER(*ldptr*).fieldname**.

The program must be loaded with the object file access routine library **libld.a**.

### See Also

---

*ldclose*(S), *ldopen*(S), *ldfcn*(F)



## ldgetname

---

retrieve symbol name for common object file symbol table entry

### Syntax

---

```
#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>
```

```
char *ldgetname (ldptr, symbol)
LDFILE *ldptr;
SYMENT *symbol;
```

### Description

---

The *ldgetname* function returns a pointer to the name associated with *symbol* as a string. The string is contained in a static buffer local to *ldgetname* that is overwritten by each call to *ldgetname*, and therefore must be copied by the caller if the name is to be saved.

The *ldgetname* function can be used to retrieve names from object files without any backward compatibility problems. The *ldgetname* function will return NULL (defined in *stdio.h*) for an object file if the name cannot be retrieved. This situation can occur:

- if the "string table" cannot be found,
- if not enough memory can be allocated for the string table,
- if the string table appears not to be a string table (for example, if an auxiliary entry is handed to *ldgetname* that looks like a reference to a name in a nonexistent string table), or
- if the name's offset into the string table is past the end of the string table.

Typically, *ldgetname* will be called immediately after a successful call to *ldtbread* to retrieve the name associated with the symbol table entry filled by *ldtbread*.

The program must be loaded with the object file access routine library *libld.a*.

## **See Also**

---

ldclose(S), ldopen(S), ldtbread(S), ldtbseek(S), ldfcn(F)

## **ldiv**

---

divides long integers

### **Syntax**

---

```
#include <stdlib.h>
```

```
struct ldiv_t
{
 long int quot; /* Quotient */
 long int rem; /* Remainder */
} ldiv(numerator, denominator)
long int numerator;
long int denominator;
```

### **Description**

---

The *ldiv* routine divides *numerator* by *denominator*, computing the quotient and the remainder. The sign of the quotient is the same as that of the mathematical quotient. Its absolute value is the largest integer which is less than the absolute value of the mathematical quotient. If the denominator is zero, the program terminates with an error message.

The *ldiv* function is similar to the *div* function, the difference being that the arguments and the members of the returned structure are all of type *long int*.

### **Return Value**

---

The *ldiv* function returns a structure of type *ldiv\_t*, comprising both the quotient and the remainder. The structure is defined in *stdlib.h*.

### **See Also**

---

`div(S)`



## Example

---

```
#include <stdlib.h>
#include <math.h>

main(argc, argv)
int argc;
char **argv;
{
 long int x,y;
 ldiv_t div_result;

 x = atol(argv[1]);
 y = atol(argv[2]);
 printf("x is %ld, y is %ld\n", x,y);

 div_result = ldiv(x,y);
 printf("The quotient is %ld, and the remainder is %ld\n",
 div_result.quot, div_result.rem);
}
```

This program takes two long integers as command line arguments and displays the results of the integer division.

## Standards Conformance

---

*ldiv* is conformant with:

ANSI X3.159-198X C Language Draft Standard, May 13, 1988.

## **ldlread, ldlnit, ldllitem**

---

manipulate line number entries of a common object file function

### **Syntax**

---

```
#include <stdio.h>
#include <filehdr.h>
#include <linenum.h>
#include <ldfcn.h>
```

```
int ldlread(ldptr, fcndidx, linenum, linent)
LDFILE *ldptr;
long fcndidx;
unsigned short linenum;
LINENO *linent;
```

```
int ldlnit(ldptr, fcndidx)
LDFILE *ldptr;
long fcndidx;
```

```
int ldllitem(ldptr, linenum, linent)
LDFILE *ldptr;
unsigned short linenum;
LINENO *linent;
```

### **Description**

---

The *ldlread* function searches the line number entries of the common object file currently associated with *ldptr*. The *ldlread* function begins its search with the line number entry for the beginning of a function and confines its search to the line numbers associated with a single function. The function is identified by *fcndidx*, the index of its entry in the object file symbol table. The *ldlread* function reads the entry with the smallest line number equal to or greater than *linenum* into the memory beginning at *linent*.

The *ldlnit* and *ldllitem* functions together perform exactly the same function as *ldlread*. After an initial call to *ldlread* or *ldlnit*, *ldllitem* may be used to retrieve a series of line number entries associated with a single function. *ldlnit* simply locates the line number entries for the function identified by *fcndidx*. *ldllitem* finds and reads the entry with the smallest line number equal to or greater than *linenum* into the memory beginning at *linent*.

The *ldlread*, *ldlinit*, and *ldlitem* functions each return either **SUCCESS** or **FAILURE**. *ldlread* will fail if there are no line number entries in the object file, if *fcnindx* does not index a function entry in the symbol table, or if it finds no line number equal to or greater than *linenum*. *ldlinit* will fail if there are no line number entries in the object file or if *fcnindx* does not index a function entry in the symbol table. *ldlitem* will fail if it finds no line number equal to or greater than *linenum*.

The programs must be loaded with the object file access routine library **libld.a**.

## See Also

---

*ldclose*(S), *ldopen*(S), *ldtbindx*(S), *ldfcn*(F)



## ldlseek, ldnlseek

---

seek to line number entries of a section of a common object file

### Syntax

---

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>
```

```
int ldlseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;
```

```
int ldnlseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;
```

### Description

---

The *ldlseek* function seeks to the line number entries of the section specified by *sectindx* of the common object file currently associated with *ldptr*.

The *ldnlseek* function seeks to the line number entries of the section specified by *sectname*.

The *ldlseek* and *ldnlseek* functions return **SUCCESS** or **FAILURE**. *ldlseek* will fail if *sectindx* is greater than the number of sections in the object file; *ldnlseek* will fail if there is no section name corresponding with *\*sectname*. Either function will fail if the specified section has no line number entries or if it cannot seek to the specified line number entries.

Note that the first section has an index of *one*.

The program must be loaded with the object file access routine library **libld.a**.

### See Also

---

ldclose(S), ldopen(S), ldshread(S), ldfcn(F)

## ldohseek

---

seek to the optional file header of a common object file

### Syntax

---

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>
```

```
int ldohseek (ldptr)
LDFILE *ldptr;
```

### Description

---

The *ldohseek* function seeks to the optional file header of the common object file currently associated with *ldptr*.

The *ldohseek* function returns **SUCCESS** or **FAILURE**. *ldohseek* will fail if the object file has no optional header or if it cannot seek to the optional header.

The program must be loaded with the **-lld** flag.

### See Also

---

ldclose(S), ldopen(S), ldhread(S), ldfcn(F)

## ldopen, ldaopen

---

open a common object file for reading

### Syntax

---

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>
```

```
LDFILE *ldopen (filename, ldptr)
char *filename;
LDFILE *ldptr;
```

```
LDFILE *ldaopen (filename, oldptr)
char *filename;
LDFILE *oldptr;
```

### Description

---

The *ldopen* and *ldclose*(S) functions are designed to provide uniform access to both simple object files and object files that are members of archive files. Thus an archive of common object files can be processed as if it were a series of simple common object files.

If *ldptr* has the value **NULL**, then *ldopen* will open *filename* and allocate and initialize the **LDFILE** structure, and return a pointer to the structure to the calling program.

If *ldptr* is valid and if **TYPE**(*ldptr*) is the archive magic number, *ldopen* will reinitialize the **LDFILE** structure for the next archive member of *filename*.

The *ldopen* and *ldclose*(S) functions are designed to work in concert. *ldclose* will return **FAILURE** only when **TYPE**(*ldptr*) is the archive magic number and there is another file in the archive to be processed. Only then should *ldopen* be called with the current value of *ldptr*. In all other cases, in particular whenever a new *filename* is opened, *ldopen* should be called with a **NULL** *ldptr* argument.



The following is a prototype for the use of *ldopen* and *ldclose* (S).

```
/* for each filename to be processed */
ldptr = NULL;
do
{
 if ((ldptr = ldopen(filename, ldptr)) != NULL)
 {
 /* check magic number */
 /* process the file */
 }
} while (ldclose(ldptr) == FAILURE);
```

If the value of *oldptr* is not **NULL**, *ldaopen* will open *filename* anew and allocate and initialize a new **LDFILE** structure, copying the **TYPE**, **OFFSET**, and **HEADER** fields from *oldptr*. *ldaopen* returns a pointer to the new **LDFILE** structure. This new pointer is independent of the old pointer, *oldptr*. The two pointers may be used concurrently to read separate parts of the object file. For example, one pointer may be used to step sequentially through the relocation information, while the other is used to read indexed symbol table entries.

Both *ldopen* and *ldaopen* open *filename* for reading. Both functions return **NULL** if *filename* cannot be opened, or if memory for the **LDFILE** structure cannot be allocated. A successful open does not ensure that the given file is a common object file or an archived object file.

The program must be loaded with the **-lld** flag.

## See Also

---

*fopen*(S), *ldclose*(S), *ldfcn*(F)

## ldrseek, ldnrseek

---

seek to relocation entries of a section of a common object file

### Syntax

---

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldrseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnrseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;
```

### Description

---

The *ldrseek* function seeks to the relocation entries of the section specified by *sectindx* of the common object file currently associated with *ldptr*.

The *ldnrseek* function seeks to the relocation entries of the section specified by *sectname*.

The *ldrseek* and *ldnrseek* functions return **SUCCESS** or **FAILURE**. *ldrseek* will fail if *sectindx* is greater than the number of sections in the object file; *ldnrseek* will fail if there is no section name corresponding with *sectname*. Either function will fail if the specified section has no relocation entries or if it cannot seek to the specified relocation entries.

Note that the first section has an index of *one*.

The program must be loaded with the **-lld** flag.

### See Also

---

ldclose(S), ldopen(S), ldshread(S), ldfcn(F)

## ldshread, ldnshread

---

read an indexed/named section header of a common object file

### Syntax

---

```
#include <stdio.h>
#include <filehdr.h>
#include <scnhdr.h>
#include <ldfcn.h>
```

```
int ldshread (ldptr, sectindx, secthead)
LDFILE *ldptr;
unsigned short sectindx;
SCNHDR *secthead;
```

```
int ldnshread (ldptr, sectname, secthead)
LDFILE *ldptr;
char *sectname;
SCNHDR *secthead;
```

### Description

---

The *ldshread* function reads the section header specified by *sectindx* of the common object file currently associated with *ldptr* into the area of memory beginning at *secthead*.

The *ldnshread* function reads the section header specified by *sectname* into the area of memory beginning at *secthead*.

The *ldshread* and *ldnshread* functions return **SUCCESS** or **FAILURE**. *ldshread* will fail if *sectindx* is greater than the number of sections in the object file; *ldnshread* will fail if there is no section name corresponding with *sectname*. Either function will fail if it cannot read the specified section header.

Note that the first section header has an index of *one*.

The program must be loaded with the **-ldd** flag.

### See Also

---

ldclose(S), ldopen(S), ldfcn(F)



## **ldsseek, ldnsseek**

---

seek to an indexed/named section of a common object file

### **Syntax**

---

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>
```

```
int ldsseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;
```

```
int ldnsseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;
```

### **Description**

---

The *ldsseek* function seeks to the section specified by *sectindx* of the common object file currently associated with *ldptr*.

The *ldnsseek* function seeks to the section specified by *sectname*.

The *ldsseek* and *ldnsseek* functions return **SUCCESS** or **FAILURE**. *ldsseek* will fail if *sectindx* is greater than the number of sections in the object file; *ldnsseek* will fail if there is no section name corresponding with *sectname*. Either function will fail if there is no section data for the specified section or if it cannot seek to the specified section.

Note that the first section has an index of *one*.

The program must be loaded with the **-lld** flag.

### **See Also**

---

ldclose(S), ldopen(S), ldshread(S), ldfcn(F)

## ldtindex

---

compute the index of a symbol table entry of a common object file

### Syntax

---

```
#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>
```

```
long ldtindex (ldptr)
LDFILE *ldptr;
```

### Description

---

The *ldtindex* function returns the (**long**) index of the symbol table entry at the current position of the common object file associated with *ldptr*.

The index returned by *ldtindex* may be used in subsequent calls to *ldtbread*(S). However, since *ldtindex* returns the index of the symbol table entry that begins at the current position of the object file, if *ldtindex* is called immediately after a particular symbol table entry has been read, it will return the index of the next entry.

The *ldtindex* function will fail if there are no symbols in the object file, or if the object file is not positioned at the beginning of a symbol table entry.

Note that the first symbol in the symbol table has an index of *zero*.

The program must be loaded with the **-lld** flag.

### See Also

---

ldclose(S), ldopen(S), ldtbread(S), ldtbseek(S), ldfcn(F)

## ldtbread

---

read an indexed symbol table entry of a common object file

### Syntax

---

```
#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>
```

```
int ldtbread (ldptr, symindex, symbol)
LDFILE *ldptr;
long symindex;
SYMENT *symbol;
```

### Description

---

The *ldtbread* function reads the symbol table entry specified by *symindex* of the common object file currently associated with *ldptr* into the area of memory beginning at **symbol**.

The *ldtbread* function returns **SUCCESS** or **FAILURE**. *ldtbread* will fail if *symindex* is greater than or equal to the number of symbols in the object file, or if it cannot read the specified symbol table entry.

Note that the first symbol in the symbol table has an index of *zero*.

The program must be loaded with the **-lld** flag.

### See Also

---

ldclose(S), ldopen(S), ldtbseek(S), ldgetname(S), ldfcn(F)



## ldtbseek

---

seek to the symbol table of a common object file

### Syntax

---

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>
```

```
int ldtbseek (ldptr)
LDFILE *ldptr;
```

### Description

---

The *ldtbseek* function seeks to the symbol table of the common object file currently associated with *ldptr*.

The *ldtbseek* function returns **SUCCESS** or **FAILURE**. *ldtbseek* will fail if the symbol table has been stripped from the object file, or if it cannot seek to the symbol table.

The program must be loaded with the **-lld** flag.

### See Also

---

ldclose(S), ldopen(S), ldtbread(S), ldfcn(F)

# libwindows

## windowing terminal function library

### Syntax

```
cc [flag ...] file ... -lwindows [library ...]
```

```
int cntlfd, fd
int chan
int origin_x, origin_y, corner_x, corner_y
char *command
```

```
cntlfd = openagent ()
```

```
chan = New (cntlfd, origin_x, origin_y, corner_x, corner_y)
```

```
chan = Newlayer (cntlfd, origin_x, origin_y, corner_x, corner_y)
```

```
fd = openchan (chan)
```

```
Runlayer (chan, command)
```

```
Current (cntlfd, chan)
```

```
Delete (cntlfd, chan)
```

```
Top (cntlfd, chan)
```

```
Bottom (cntlfd, chan)
```

```
Move (cntlfd, chan, origin_x, origin_y)
```

```
Reshape (cntlfd, chan, origin_x, origin_y, corner_x, corner_y)
```

```
Exit (cntlfd)
```

### Description

This library of routines enables a program running on a host system to perform windowing terminal functions (see *layers(C)*).

The **openagent()** routine opens the control channel of the *xt(HW)* channel group to which the calling process belongs. Upon successful completion, *openagent()* returns a file descriptor, *cntlfd*, that can be passed to any of the other *libwindows* routines except *openchan()* and *Runlayer()*. (*cntlfd* can also be passed to *close(S)*.) Otherwise, the value **-1** is returned.

The *New()* routine creates a new layer with a separate shell. The *origin\_x*, *origin\_y*, *corner\_x*, and *corner\_y* arguments are the coordinates of the layer rectangle. If all the coordinate arguments are 0, the user must define the layer's rectangle interactively. The layer appears on top of any overlapping layers. The layer is not made current (that is, the keyboard is not attached to the new layer). Upon successful

completion, *New()* returns the *xt(HW)* channel number associated with the layer. Otherwise, the value **-1** is returned.

The *Newlayer()* routine creates a new layer without executing a separate shell. Otherwise it is identical to *New()*, described above.

The *openchan()* routine opens the channel argument *chan* which is obtained from the *New()* or *Newlayer()* routine. Upon successful completion, *openchan()* returns a file descriptor that can be used as input to *write(S)* or *close(S)*. Otherwise, the value **-1** is returned.

The *Runlayer()* routine runs the specified *command* in the layer associated with the channel argument *chan*. Any processes currently attached to this layer will be killed, and the new process will have the environment of the *layers(C)* process.

The *Current()* routine makes the layer associated with the channel argument *chan* current (i.e., attached to the keyboard).

The *Delete()* routine deletes the layer associated with the channel argument *chan* and kills all host processes associated with the layer.

The *Top()* routine makes the layer associated with the channel argument *chan* appear on top of all overlapping layers.

The *Bottom()* routine puts the layer associated with the channel argument *chan* under all overlapping layers.

The *Move()* routine moves the layer associated with the channel argument *chan* from its current screen location to a new screen location at the origin point (*origin\_x*, *origin\_y*). The size and contents of the layer are maintained.

The *Reshape()* routine reshapes the layer associated with the channel argument *chan*. The arguments *origin\_x*, *origin\_y*, *corner\_x*, and *corner\_y* are the new coordinates of the layer rectangle. If all the coordinate arguments are 0, the user is allowed to define the layer's rectangle interactively.

The *Exit()* routine causes the *layers(C)* program to exit, killing all processes associated with it.

## Return Value

---

Upon successful completion, *Runlayer()*, *Current()*, *Delete()*, *Top()*, *Bottom()*, *Move()*, *Reshape()*, and *Exit()* return a **0**, while *openagent()*, *New()*, *Newlayer()*, and *openchan()* return values as described above under each routine. If an error occurs, **-1** is returned.



## Files

---

/usr/lib/libwindows.a      windowing terminal function library

## See Also

---

close(S), jagent(M), write(S)  
layers(C), xt(HW).

## Note

---

The values of layer rectangle coordinates are dependent on the type of terminal. This dependency affects the routines that pass layer rectangle coordinates: *Move()*, *New()*, *Newlayer()*, and *Reshape()*. Some terminals will expect these numbers to be passed as character positions (bytes); others will expect the information to be in pixels (bits).

For example, for the AT&T TELETYPE 5620 DMD terminal, *New()*, *Newlayer()*, and *Reshape()* take minimum values of 8 (pixels) for *origin\_x* and *origin\_y* and maximum values of 792 (pixels) for *corner\_x* and 1016 (pixels) for *corner\_y*. In addition, the minimum layer size is 28 by 28 pixels and the maximum layer size is 784 by 1008 pixels.

# link

---

link to a file

## Syntax

---

```
int link (path1, path2)
char *path1, *path2;
```

## Description

---

The *path1* argument points to a path name naming an existing file. The *path2* argument points to a path name naming the new directory entry to be created. The *link* system call creates a new link (directory entry) for the existing file.

The *link* system call will fail and no link will be created if one or more of the following is true:

- |           |                                                                                                                    |
|-----------|--------------------------------------------------------------------------------------------------------------------|
| [ENOTDIR] | A component of either path prefix is not a directory.                                                              |
| [ENOENT]  | A component of either path prefix does not exist.                                                                  |
| [EACCES]  | A component of either path prefix denies search permission.                                                        |
| [ENOENT]  | The file named by <i>path1</i> does not exist.                                                                     |
| [EEXIST]  | The link named by <i>path2</i> exists.                                                                             |
| [EPERM]   | The file named by <i>path1</i> is a directory and the effective user ID is not super-user.                         |
| [EXDEV]   | The link named by <i>path2</i> and the file named by <i>path1</i> are on different logical devices (file systems). |
| [ENOENT]  | <i>path2</i> points to a null path name.                                                                           |
| [EACCES]  | The requested link requires writing in a directory with a mode that denies write permission.                       |
| [EROFS]   | The requested link requires writing in a directory on a read-only file system.                                     |
| [EFAULT]  | <i>path</i> points outside the allocated address space of the process.                                             |

LINK (S)

LINK (S)

|             |                                                                                          |
|-------------|------------------------------------------------------------------------------------------|
| [EMLINK]    | The maximum number of links to a file would be exceeded.                                 |
| [EINTR]     | A signal was caught during the <i>link</i> system call.                                  |
| [ENOLINK]   | <i>Path</i> points to a remote machine and the link to that machine is no longer active. |
| [EMULTIHOP] | Components of <i>path</i> require hopping to multiple remote machines.                   |
| [ENOSPC]    | The directory containing the link cannot be extended.                                    |

## See Also

---

unlink(S)

## Diagnostics

---

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## Standards Conformance

---

*link* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;  
and NIST FIPS 151-1.



## lock

---

locks a process in primary memory

### Syntax

---

```
int lock(flag);
int flag;
```

### Description

---

If the *flag* argument is nonzero, the process executing this call will not be swapped except if it is required to grow. If the argument is zero, the process is *unlocked*. This call may only be executed by the super-user.

### Notes

---

*locked* processes interfere with the compaction of primary memory and can cause deadlock. Systems with small memory configurations should avoid using this call. It is best to lock process soon after booting because that will tend to lock them into one end of memory.

This feature is a XENIX specific enhancement and may not be present in all UNIX implementations. This routine must be linked using the linker option **-lx**.

### Standards Conformance

---

*lock* is conformant with:

The X/Open Portability Guide II of January 1987.

## lockf

---

record locking on files

### Syntax

---

```
#include <unistd.h>

int lockf (fildes, function, size)
long size;
int fildes, function;
```

### Description

---

The *lockf* command will allow sections of a file to be locked; (advisory or mandatory write locks are used depending on the mode bits of the file [see *chmod(S)*]). Locking calls from other processes which attempt to lock the locked file section will either return an error value or be put to sleep until the resource becomes unlocked. All the locks for a process are removed when the process terminates. (See *fcntl(S)* for more information about record locking.)

*fildes* is an open file descriptor. The file descriptor must have *O\_WRONLY* or *O\_RDWR* permission in order to establish a lock with this function call.

*function* is a control value which specifies the action to be taken. The permissible values for *function* are defined in *<unistd.h>* as follows:

```
#define F_ULOCK 0 /* Unlock a previously locked section */
#define F_LOCK 1 /* Lock a section for exclusive use */
#define F_TLOCK 2 /* Test and lock a section for exclusive use */
#define F_TEST 3 /* Test section for other processes locks */
```

All other values of *function* are reserved for future extensions and will result in an error return if not implemented.

*F\_TEST* is used to detect if a lock by another process is present on the specified section. *F\_LOCK* and *F\_TLOCK* both lock a section of a file if the section is available. *F\_ULOCK* removes locks from a section of the file.

*size* is the number of contiguous bytes to be locked or unlocked. The section to be locked starts at the current offset in the file and extends forward for a positive size and backward for a negative size (the preceding bytes up to but not including the current offset). If *size* is zero, the section from the current offset through the largest file offset is locked (that is, from the current offset through the present or any future end-of-file). An area need not be allocated to the file in order to be locked as such locks may exist past the end-of-file.



The sections locked with F\_LOCK or F\_TLOCK may, in whole or in part, contain or be contained by a previously locked section for the same process. When this occurs, or if adjacent sections occur, the sections are combined into a single section. If the request requires that a new element be added to the table of active locks and this table is already full, an error is returned, and the new section is not locked.

F\_LOCK and F\_TLOCK requests differ only by the action taken if the resource is not available. F\_LOCK will cause the calling process to sleep until the resource is available. F\_TLOCK will cause the function to return a -1 and set *errno* to [EACCES] if the section is already locked by another process.

F\_ULOCK requests may, in whole or in part, release one or more locked sections controlled by the process. When sections are not fully released, the remaining sections are still locked by the process. Releasing the center section of a locked section requires an additional element in the table of active locks. If this table is full, an [EDEADLK] error is returned, and the requested section is not released.

A potential for deadlock occurs if a process controlling a locked resource is put to sleep by accessing another process's locked resource. Thus calls to *lockf* or *fcntl* scan for a deadlock prior to sleeping on a locked resource. An error return is made if sleeping on the locked resource would cause a deadlock.

Sleeping on a resource is interrupted with any signal. The *alarm(S)* command may be used to provide a timeout facility in applications which require this facility.

The *lockf* utility will fail if one or more of the following is true:

- |           |                                                                                                                                                                                                         |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [EBADF]   | <i>fildev</i> is not a valid open descriptor.                                                                                                                                                           |
| [EACCES]  | <i>cmd</i> is F_TLOCK or F_TEST and the section is already locked by another process.                                                                                                                   |
| [EDEADLK] | <i>cmd</i> is F_LOCK and a deadlock would occur. Also the <i>cmd</i> is either F_LOCK, F_TLOCK, or F_ULOCK and the number of entries in the lock table would exceed the number allocated on the system. |
| [ECOMM]   | <i>fildev</i> is on a remote machine and the link to that machine is no longer active.                                                                                                                  |

## See Also

---

chmod(S), close(S), creat(S), fcntl(S), intro(S), open(S), read(S), write(S)



## Diagnostics

---

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## Warnings

---

Unexpected results may occur in processes that do buffering in the user address space. The process may later read/write data which is/was locked. The standard I/O package is the most common source of unexpected buffering.

Because in the future the variable *errno* will be set to EAGAIN rather than EACCES when a section of a file is already locked by another process, portable application programs should expect and test for either value.

## Standards Conformance

---

*lockf* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

# locking

---

locks or unlocks a file region for reading or writing

## Syntax

---

```
#include <sys/types.h>
#include <sys/locking.h>
```

```
int locking(fildes, mode, size);
int fildes, mode;
long size;
```

## Description

---

*locking* allows a specified number of bytes in a file to be controlled by the locking process. Other processes which attempt to read or write a portion of the file containing the locked region may sleep until the area becomes unlocked depending upon the mode in which the file region was locked.

A file must be open with read or read/write permission for a read lock to be performed. Write or read/write permission is required for a write lock. If either of these conditions are not met, the lock will fail with the error EINVAL.

A process that attempts to write to or read a file region that has been locked against reading and writing by another process (using the LK\_LOCK or LK\_NBLCK mode) will sleep until the region of the file has been released by the locking process.

A process that attempts to write to a file region that has been locked against writing by another process (using the LK\_RLCK or LK\_NBRLOCK mode) will sleep until the region of the file has been released by the locking process, but a read request for that file region will proceed normally.

A process that attempts to lock a region of a file that contains areas that have been locked by other processes will sleep if it has specified the LK\_LOCK or LK\_RLCK mode in its lock request, but will return with the error EACCES if it specified LK\_NBLCK or LK\_NBRLOCK.

*fildes* is the value returned from a successful *creat*, *open*, *dup*, or *pipe* system call.

*mode* specifies the type of lock operation to be performed on the file region. The available values for *mode* are:

#### LK\_UNLCK 0

Unlocks the specified region. The calling process releases a region of the file it had previously locked.

#### LK\_LOCK 1

Locks the specified region. The calling process will sleep until the entire region is available if any part of it has been locked by a different process. The region is then locked for the calling process and no other process may read or write in any part of the locked region. (lock against read and write).

#### LK\_NBLCK 2

Locks the specified region. If any part of the region is already locked by a different process, return the error EACCES instead of waiting for the region to become available for locking (nonblocking lockrequest).

#### LK\_RLCK 3

Same as LK\_LOCK except that the locked region may be read by other processes (read permitted lock).

#### LK\_NBRLCK 4

Same as LK\_NBLCK except that the locked region may be read by other processes (nonblocking, read permitted lock).

The *locking* utility uses the current file pointer position as the starting point for the *locking* of the file segment. So a typical sequence of commands to *lock* a specific range within a file might be as follows:

```
fd=open("datafile",O_RDWR);
lseek(fd, 200L, 0);
locking(fd, LK_LOCK, 200L);
```

Accordingly, to *lock* or *unlock* an entire file a *seek* to the beginning of the file (position 0) must be done and then a *locking* call must be executed with a size of 0.

*size* is the number of contiguous bytes to be locked or unlocked. The region to be locked starts at the current offset in the file. If *size* is 0, the entire file (up to a maximum of 2 to the power of 30 bytes) is locked or unlocked. *size* may extend beyond the end of the file, in which case only the process issuing the lock call may access or add information to the file within the boundary defined by *size*.



The potential for a deadlock occurs when a process controlling a locked area is put to sleep by accessing another process' locked area. Thus calls to *locking*, *read*, or *write* scan for a deadlock prior to sleeping on a locked region. An EDEADLK (or EDEADLOCK) error return is made if sleeping on the locked region would cause a deadlock.

Lock requests may, in whole or part, contain or be contained by a previously locked region for the same process. When this occurs, or when adjacent regions are locked, the regions are combined into a single area if the mode of the lock is the same (i.e.; either read permitted or regular lock). If the mode of the overlapping locks differ, the locked areas will be assigned assuming that the *most recent request* must be satisfied. Thus if a read only lock is applied to a region, or part of a region, that had been previously locked by the same process against both reading and writing, the area of the file specified by the new lock will be locked for read only, while the remaining region, if any, will remain locked against reading and writing. There is no arbitrary limit to the number of regions which may be locked in a file. There is however a system-wide limit on the total number of locked regions. This limit is 200 for UNIX style systems.

Unlock requests may, in whole or part, release one or more locked regions controlled by the process. When regions are not fully released, the remaining areas are still locked by the process. Release of the center section of a locked area requires an additional locked element to hold the separated section. If the lock table is full, an error is returned, and the requested region is not released. Only the process which locked the file region may unlock it. An unlock request for a region that the process does not have locked, or that is already unlocked, has no effect. When a process terminates, all locked regions controlled by that process are unlocked.

If a process has done more than one open on a file, *all* locks put on the file by that process will be released on the first close of the file.

Although no error is returned if locks are applied to special files or pipes, read/write operations on these types of files will ignore the locks. Locks may not be applied to a directory.

## See Also

---

`creat(S)`, `open(S)`, `read(S)`, `write(S)`, `dup(S)`, `close(S)`, `lseek(S)`

## Diagnostics

---

*locking* returns the value (int) -1 if an error occurs. If any portion of the region has been locked by another process for the LK\_LOCK and LK\_RLCK actions and the lock request is to test only, *errno* is set to EAGAIN when used with UNIX System V binaries. If the binary using this routine is a UNIX 3.0 binary, this *errno* is set to EACCES. If the file specified is a directory, *errno* is set to EACCES. If locking the region would cause a deadlock, *errno* is set to EDEADLK (or EDEADLOCK). If there are no more free internal locks, *errno* is set to EDEADLK (or EDEADLOCK).

## Notes

---

This routine must be linked with the linker option **-lx**.

## logname

---

return login name of user

### Syntax

---

`char *logname( )`

### Description

---

The *logname* function returns a pointer to the null-terminated login name; it extracts the **LOGNAME** environment variable from the user's environment.

This routine is kept in **/lib/libPW.a** and must be linked with the **-lPW** flag.

### Files

---

**/etc/profile**

### See Also

---

**getenv(S)**, **profile(M)**, **environ(M)**, **env(C)**, **login(M)**.

### Notes

---

The return values point to static data whose content is overwritten by each call.

This method of determining a login name is subject to forgery.

### Standards Conformance

---

*logname* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



# **lsearch, lfind**

linear search and update

---

## **Syntax**

---

```
#include <stdio.h>
```

```
#include <search.h>
```

```
void *lsearch ((void *)key, (char *)base, nelp, sizeof(*key), compar)
```

```
unsigned *nelp;
```

```
int (*compar)();
```

```
void *lfind ((void *)key, (char *)base, nelp, sizeof(*key), compar)
```

```
unsigned *nelp;
```

```
int (*compar)();
```

## **Description**

---

The *lsearch* function is a linear search routine generalized from Knuth (6.1) Algorithm S. It returns a pointer into a table indicating where a datum may be found. If the datum does not occur, it is added at the end of the table. **key** points to the datum to be sought in the table. **base** points to the first element in the table. **nelp** points to an integer containing the current number of elements in the table. The integer is incremented if the datum is added to the table. **compar** is the name of the comparison function which the user must supply (*strcmp*, for example). It is called with two arguments that point to the elements being compared. The function must return zero if the elements are equal and non-zero otherwise.

*lfind* is the same as *lsearch* except that if the datum is not found, it is not added to the table. Instead, a NULL pointer is returned.

## Example

---

This fragment will read in less than TABSIZE strings of length less than ELSIZE and store them in a table, eliminating duplicates.

```
#include <stdio.h>
#include <search.h>

#define TABSIZE 50
#define ELSIZE 120

char line[ELSIZE], tab[TABSIZE][ELSIZE], *lsearch();
unsigned nel = 0;
int strcmp();
...
while (fgets(line, ELSIZE, stdin) != NULL &&
 nel < TABSIZE)
 (void) lsearch(line, (char *)tab, &nel,
 ELSIZE, strcmp);
...
```

## See Also

---

bsearch(S), hsearch(S), string(S), tsearch(S)

## Diagnostics

---

If the searched-for datum is found, both *lsearch* and *lfind* return a pointer to it. Otherwise, *lfind* returns NULL and *lsearch* returns a pointer to the newly added element.

## Notes

---

The pointers to the key and the element at the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

Although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

Undefined results can occur if there is not enough room in the table to add a new item.

## **Standards Conformance**

---

*lfind* and *lsearch* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



# **lseek**

---

move read/write file pointer

## **Syntax**

---

```
#include <unistd.h>
```

```
long lseek (fildes, offset, whence)
int fildes;
long offset;
int whence;
```

## **Description**

---

The *fildes* argument is a file descriptor returned from a *creat*, *open*, *dup*, or *fcntl* system call. The *lseek* system call sets the file pointer associated with *fildes* as follows:

If *whence* is 0, the pointer is set to *offset* bytes.

If *whence* is 1, the pointer is set to its current location plus *offset*.

If *whence* is 2, the pointer is set to the size of the file plus *offset*.

Symbolic constants for *whence* are defined in the <unistd.h> header file:

| <i>Name</i> | <i>Description</i>                                        |
|-------------|-----------------------------------------------------------|
| SEEK_SET    | Set file-pointer equal to <i>offset</i> bytes.            |
| SEEK_CUR    | Set file-pointer to current location plus <i>offset</i> . |
| SEEK_END    | Set file-pointer to EOF plus <i>offset</i> .              |

Upon successful completion, the resulting pointer location, as measured in bytes from the beginning of the file, is returned. Note that if *fildes* is a remote file descriptor and *offset* is negative, *lseek* will return the file pointer even if it is negative.

*lseek* will fail and the file pointer will remain unchanged if one or more of the following is true:

|          |                                                  |
|----------|--------------------------------------------------|
| [EBADF]  | <i>fildes</i> is not an open file descriptor.    |
| [ESPIPE] | <i>fildes</i> is associated with a pipe or fifo. |

[EINVAL and SIGSYS signal]

*whence* is not 0, 1, or 2.

[EINVAL]

*fildev* is not a remote file descriptor, and the resulting file pointer would be negative.

Some devices are incapable of seeking. The value of the file pointer associated with such a device is undefined.

## See Also

---

`creat(S)`, `dup(S)`, `fcntl(S)`, `open(S)`

## Diagnostics

---

Upon successful completion, a non-negative integer indicating the file pointer value is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## Standards Conformance

---

*lseek* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

# malloc, free, realloc, calloc

---

allocates main memory

## Syntax

---

**char \*malloc(size)**  
unsigned size;

**void free(ptr)**  
**char \*ptr;**

**char \*realloc(ptr, size)**  
**char \*ptr;**  
unsigned size;

**char \*calloc(nelem, elsize)**  
unsigned nelem, elsize;

## Description

---

There are two versions of the *malloc(S)* package. Both versions are documented in these *malloc(S)* manual pages; the description for the other package starts on page 3. This portion of the manual page documents the standard, default *malloc(S)* package. This version of *malloc* and *free* provide a simple general-purpose memory allocation package. *malloc* returns a pointer to a block of at least *size* bytes beginning on a word boundary.

The argument to *free* is a pointer to a block previously allocated by *malloc*; this space is made available for further allocation, but its contents are left undisturbed.

Undefined results will occur if space assigned by *malloc* is overrun or if some random number is handed to *free*.

*malloc* allocates the first contiguous reach of free space found in a circular search from the last block allocated or freed, coalescing adjacent free blocks as it searches. It calls *sbrk* (see *sbrk(S)*) to get more memory from the system when there is no suitable space already free.

*realloc* changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes. If no free block of *size* bytes is available in the storage arena, then *realloc* will ask *malloc* to enlarge the arena by *size* bytes and will then move the data to the new space.



*realloc* also works if *ptr* points to a block freed since the last call of *malloc*, *realloc*, or *calloc*; thus sequences of *free*, *malloc* and *realloc* can exploit the search strategy of *malloc* to do storage compaction.

*calloc* allocates space for an array of *nelem* elements of size *elsize*. The space is initialized to zeros.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

## See Also

---

brkctl(S), malloc(S), sbrk(S)

## Diagnostics

---

*malloc*, *realloc* and *calloc* return a null pointer (0) if there is no available memory or if the area has been detectably corrupted by storing outside the bounds of a block. When *realloc* returns 0, the block pointed to by *ptr* may be destroyed.

## Note

---

As noted, *malloc* calls *sbrk* to allocate memory. Since *sbrk* takes a signed integer as its argument, *malloc* will fail if an attempt is made to allocate more memory than a signed integer will hold (32K -1).

Search time increases when many objects have been allocated; that is, if a program allocates but never frees, then each successive allocation takes longer. For an alternate and more flexible implementation see the *malloc* (S) documented on pages 3-5 of this manual entry.

# malloc, free, realloc, calloc, malloc, mallinfo

---

allocates main memory quickly

## Syntax

---

```
#include <malloc.h>
```

```
char *malloc (size)
unsigned size;
```

```
void free (ptr)
char *ptr;
```

```
char *realloc (ptr, size)
char *ptr;
unsigned size;
```

```
char *calloc (nelem, elsize)
unsigned nelem, elsize;
```

```
int malloc (cmd, value)
int cmd, value;
```

```
struct mallinfo mallinfo()
```

## Description

---

There are two versions of the *malloc(S)* package. This is the library version which provides a simple general-purpose memory allocation package, that runs considerably faster than the other *malloc(S)* package. Both versions are documented in these *malloc(S)* manual pages; the description of the standard default package starts on page 1.

This *malloc(S)* package is found in the library “malloc” and is loaded when the option **-lmalloc** is used with *cc*(CP) or *ld*(CP).

*malloc* returns a pointer to a block of at least *size* bytes suitably aligned for any use.

The argument to *free* is a pointer to a block previously allocated by *malloc*; after *free* is performed this space is made available for further allocation, and its contents destroyed (see *malloc* below for a way to change this behavior).

Undefined results occur if the space assigned by *malloc* is overrun or if some random number is handed to *free*.

*realloc* changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes.

*calloc* allocates space for an array of *nelem* elements of size *elsize*. The space is initialized to zeros.

*mallopt* provides for control over the allocation algorithm. The available values for *cmd* are:

#### M\_MXFAST

Set *maxfast* to *value*. The algorithm allocates all blocks below the size of *maxfast* in large groups and then does them out very quickly. The default value for *maxfast* is 0.

#### M\_NLBLKS

Set *numlblks* to *value*. The above mentioned "large groups" each contain *numlblks* blocks. *numlblks* must be greater than 0. The default value for *numlblks* is 100.

M\_GRAIN Set *grain* to *value*. The sizes of all blocks smaller than *maxfast* are considered to be rounded up to the nearest multiple of *grain*. *grain* must be greater than 0. The default value of *grain* is the smallest number of bytes which will allow alignment of any data type. *value* will be rounded up to a multiple of the default when *grain* is set.

M\_KEEP Preserve data in a freed block until the next *malloc*, *realloc*, or *calloc*. This option is provided only for compatibility with the old version of *malloc* and is not recommended.

These values are defined in the `<malloc.h>` header file.

*mallopt* may be called repeatedly, but may not be called after the first small block is allocated.

*mallinfo* provides instrumentation describing space usage. It returns the structure:



```

struct mallinfo {
 int arena; /* total space in arena */
 int ordblks; /* number of ordinary blocks */
 int smblks; /* number of small blocks */
 int hblkhds; /* space in holding block headers */
 int hblks; /* number of holding blocks */
 int usmblks; /* space in small blocks in use */
 int fsmblks; /* space in free small blocks */
 int uordblks; /* space in ordinary blocks in use */
 int fordblks; /* space in free ordinary blocks */
 int keepcost; /* space penalty if keep option */
 /* is used */
}

```

This structure is defined in the `<malloc.h>` header file.

Here is an example program code segment for the **mallinfo** function:

```

#include <stdio.h>
#include <malloc.h>

main()
{
 char *malloc, *cp;
 struct mallinfo minfo;

 if ((cp = malloc(1024)) == NULL)
 {
 perror("Malloc");
 exit(1);
 }

 minfo = mallinfo();
 printf("%d %d %d0", minfo.arena, minfo.ordblks, minfo.uordblks);
}

```

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

## See Also

---

*Programmer's Guide*

brkctl(S), malloc(S), sbrk(S)

## Diagnostics

---

*malloc*, *realloc* and *calloc* return a NULL pointer if there is not enough available memory. When *realloc* returns NULL, the block pointed to by *ptr* is left intact. If *mallopt* is called after any allocation or if *cmd* or *value* are invalid, non-zero is returned. Otherwise, it returns zero.

## Warnings

---

This package usually uses more data space than the other *malloc*(S).

The code size is also bigger than the other *malloc*(S).

Note that unlike the other *malloc*(S), this package does not preserve the contents of a block when it is freed, unless the M\_KEEP option of *mallopt* is used.

Undocumented features of the other *malloc*(S) have not been duplicated.

These routines must be linked with the **-lmalloc** linker option.

## Standards Conformance

---

*calloc*, *free*, *malloc* and *realloc* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

*mallinfo* and *mallopt* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

and The X/Open Portability Guide II of January 1987.

# matherr

## error-handling function

### Syntax

```
#include <math.h>
```

```
int matherr (x)
struct exception *x;
```

### Description

The *matherr* function is invoked by functions in the Math Library when errors are detected. Users may define their own procedures for handling errors by including a function named *matherr* in their programs. The *matherr* function must be of the form described above. When an error occurs, a pointer to the exception structure *x* will be passed to the user-supplied *matherr* function. This structure, which is defined in the <math.h> header file, is as follows:

```
struct exception {
 int type;
 char *name;
 double arg1, arg2, retval;
};
```

The element *type* is an integer describing the type of error that has occurred, from the following list of constants (defined in the header file):

|           |                              |
|-----------|------------------------------|
| DOMAIN    | argument domain error        |
| SING      | argument singularity         |
| OVERFLOW  | overflow range error         |
| UNDERFLOW | underflow range error        |
| TLOSS     | total loss of significance   |
| PLOSS     | partial loss of significance |

The element *name* points to a string containing the name of the function that incurred the error. The variables *arg1* and *arg2* are the arguments with which the function was invoked. *retval* is set to the default value that will be returned by the function unless the user's *matherr* sets it to a different value.

If the user's *matherr* function returns non-zero, no error message will be printed, and *errno* will not be set.



If *matherr* is not supplied by the user, the default error-handling procedures, described with the math functions involved, will be invoked upon error. These procedures are also summarized in the table below. In every case, *errno* is set to EDOM or ERANGE and the program continues.

## Example

---

```
#include <math.h>

int
matherr(x)
register struct exception *x;
{
 switch (x->type) {
 case DOMAIN:
 /* change sqrt to return sqrt(-arg1), not 0 */
 if (!strcmp(x->name, "sqrt")) {
 x->retval = sqrt(-x->arg1);
 return (0); /* print message and set errno */
 }
 case SING:
 /* all other domain or sing errors, print message and abort */
 fprintf(stderr, "domain error in %s\n", x->name);
 abort ();
 case PLOSS:
 /* print detailed error message */
 fprintf(stderr, "loss of significance in %s(%g) = %g\n",
 x->name, x->arg1, x->retval);
 return (C); /* take no other action */
 }
 return (0); /* all other errors, execute default procedure */
}
```

| DEFAULT ERROR HANDLING PROCEDURES |                        |       |          |           |        |        |
|-----------------------------------|------------------------|-------|----------|-----------|--------|--------|
|                                   | <i>Types of Errors</i> |       |          |           |        |        |
| type                              | DOMAIN                 | SING  | OVERFLOW | UNDERFLOW | TLOSS  | PLOSS  |
| <i>errno</i>                      | EDOM                   | EDOM  | ERANGE   | ERANGE    | ERANGE | ERANGE |
| BESSEL:                           | -                      | -     | -        | -         | M, 0   | *      |
| y0, y1, yn (arg ≤ 0)              | M, -H                  | -     | -        | -         | -      | -      |
| EXP:                              | -                      | -     | H        | 0         | -      | -      |
| LOG, LOG10:                       |                        |       |          |           |        |        |
| (arg < 0)                         | M, -H                  | -     | -        | -         | -      | -      |
| (arg = 0)                         | -                      | M, -H | -        | -         | -      | -      |
| POW:                              | -                      | -     | ±H       | 0         | -      | -      |
| neg ** non-int                    | M, 0                   | -     | -        | -         | -      | -      |
| 0 ** non-pos                      |                        |       |          |           |        |        |
| SQRT:                             | M, 0                   | -     | -        | -         | -      | -      |
| GAMMA:                            | -                      | M, H  | H        | -         | -      | -      |
| HYPOT:                            | -                      | -     | H        | -         | -      | -      |
| SINH:                             | -                      | -     | ±H       | -         | -      | -      |
| COSH:                             | -                      | -     | H        | -         | -      | -      |
| SIN, COS, TAN: -                  | -                      | -     | -        | M, 0      | *      |        |
| ASIN, ACOS, ATAN2: M, 0           | -                      | -     | -        | -         | -      |        |

|                |    |                                               |
|----------------|----|-----------------------------------------------|
| ABBREVIATIONS: | *  | As much as possible of the value is returned. |
|                | M  | Message is printed (EDOM error).              |
|                | H  | HUGE is returned.                             |
|                | -H | -HUGE is returned.                            |
|                | ±H | HUGE or -HUGE is returned.                    |
|                | 0  | 0 is returned.                                |

## Standards Conformance

*matherr* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.





Using *memmove*, string *Source* is copied into string *Target*. *sizeof* returns the size of the string, including the end-of-string character, effectively shortening *Target*.

## Standards Conformance

---

*memmove* is conformant with:

ANSI X3.159-198X C Language Draft Standard, May 13, 1988.

# memory: memccpy, memchr, memcmp, memcpy, memset

---

## memory operations

### Syntax

---

```
#include <memory.h>
```

```
char *memccpy (s1, s2, c, n)
char *s1, *s2;
int c, n;
```

```
void *memchr (s, c, n)
const void *s;
int c;
size_t n;
```

```
int memcmp (s1, s2, n)
char *s1, *s2;
int n;
```

```
void *memcpy (s1, s2, n)
void *s1;
const *s2;
size_t n;
```

```
void *memset (s, c, n)
void *s;
int c;
size_t n;
```

### Description

---

These functions operate as efficiently as possible on memory areas (arrays of characters bounded by a count, not terminated by a null character). They do not check for the overflow of any receiving memory area.

*memccpy* copies characters from memory area *s2* into *s1*, stopping after the first occurrence of character *c* has been copied, or after *n* characters have been copied, whichever comes first. It returns a pointer to the character after the copy of *c* in *s1*, or a NULL pointer if *c* was not found in the first *n* characters of *s2*.

*memchr* returns a pointer to the first occurrence of character *c* in the first *n* characters of memory area *s*, or a NULL pointer if *c* does not occur.

*memcmp* compares its arguments, looking at the first *n* characters only, and returns an integer less than, equal to, or greater than 0, according as *s1* is lexicographically less than, equal to, or greater than *s2*.

*memcpy* copies *n* characters from memory area *s2* to *s1*. It returns *s1*.

*memset* sets the first *n* characters in memory area *s* to the value of character *c*. It returns *s*.

For user convenience, all these functions are declared in the optional <memory.h> header file.

## Notes

---

*memcmp* is implemented by using the most natural character comparison on the machine. Thus the sign of the value returned when one of the characters has its high order bit set is not the same in all implementations and should not be relied upon.

Character movement is performed differently in different implementations. Thus overlapping moves may yield surprises.

## Standards Conformance

---

*memcpy* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

*memchr*, *memcmp*, *memcpy* and *memset* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
and ANSI X3.159-198X C Language Draft Standard, May 13, 1988.



## menu

---

### CRT menu routines

### Syntax

---

```
#include <menu.h>
```

```
cc [flags] files -lmenu -lcurses [libraries]
```

```
MENU *new_menu(ip)
ITEM ** ip;
```

```
int free_menu(m)
MENU *m;
```

```
int set_menu_items(m,i)
MENU *m;
ITEM ** i;
```

```
ITEM ** menu_items(m)
MENU * m;
```

```
int set_menu_format(m, c, r)
MENU *m;
int c, r;
```

```
void menu_format(m, rp, cp)
MENU *m;
int *rp, *cp;
```

```
int set_menu_mark(m, n)
MENU *m;
char *n;
```

```
char *menu_mark(m)
MENU *m;
```

```
int scale_menu(m, rp, cp)
MENU *m;
int *rp, *cp;
```

```
int set_menu_win(m, w)
MENU *m;
WINDOW *w;
```

```
WINDOW *menu_win(m)
MENU *m;
```

```
int set_menu_sub(m, w)
```

## MENU (S)

```
MENU *m;
WINDOW *w;
```

```
WINDOW *menu_sub(m)
MENU *m;
```

```
int set_menu_fore(m, c)
MENU *m;
int c;
```

```
int menu_fore(m)
MENU *m;
```

```
int set_menu_back(m, c)
MENU *m;
int c;
```

```
int menu_back(m)
MENU *m;
```

```
int set_menu_grey(m, c)
MENU *m;
int c;
```

```
int menu_grey(m)
MENU *m;
```

```
int set_menu_pad(m, c)
MENU *m;
int c;
```

```
int menu_pad(m)
MENU *m;
```

```
int post_menu(m)
MENU *m;
```

```
int unpost_menu(m)
MENU *m;
```

```
int menu_driver(m, c)
MENU *m;
int c;
```

```
int set_item_init(m, f)
MENU *m;
ITEM *item;
PTF_void f;
```

```
PTF_void item_init(m)
MENU *m;
```

## MENU (S)

```
set_item_term(m, f)
MENU *m;
PTF_void f;

PTF_void item_term(m)
MENU *m;

int set_menu_init(m, f)
MENU *m;
PTF_void f;

PTF_void menu_init(m)
MENU *m;

int set_menu_term(m, f)
MENU *m;
PTF_void f;

PTF_void menu_term(m)
MENU *m;

int set_current_item(m, i)
MENU *m;
ITEM *item;

ITEM *current_item(m)
MENU *m;

int item_index(i)
ITEM *i;

int set_top_row(m, c)
MENU *m;
int *c;

int top_row(m)
MENU *m;

int pos_menu_cursor(m)
MENU *m;

int set_menu_pattern(m, n)
MENU *m;
char *n;

char *menu_pattern(m)
MENU *m;

int set_menu_userptr(m, n)
MENU *m;
chr *n;
```



```
char *menu_userptr(m)
MENU *m;
```

```
int set_menu_opts(m, o)
MENU *m;
OPTIONS o;
```

```
OPTIONS menu_opts(m)
MENU *m;
```

```
int menu_opts_on (m, o)
MENU *m;
OPTIONS o;
```

```
int menu_opts_off (m, o)
MENU *m;
OPTIONS o;
```

## Description

---

These routines allow you to create, display, and access menus. Menus can be displayed on any display device supported by the low-level () library **curses**(S). Once you compile your program **including** the MENU header file **menu.h**, you should link it with the MENU and **curses** library routines.

## Functions

---

The following is a list of MENU routines.

*new\_menu(ip)* creates a new menu connected to the given item pointer array and returns a pointer to the new menu.

*free\_menu(m)* disconnects the menu from its associated item pointer array and free the storage allocated for the menu.

*set\_menu\_items(m,i)* changes the item pointer array connected to the given menu to item pointer array *i*.

*menu\_items(m)* returns a pointer to the item pointer array connected to menu *m*.

*set\_menu\_format(m,c, r)* sets the maximum number of rows and columns of items that may be displayed at one time on a menu.

*menu\_format(m,rp, cp)* returns the maximum number of rows and columns that may be displayed at one time on a menu. *rp* and *cp* are pointers to the values used to return these numbers.

The mark string distinguishes selected items in a multi-valued menu and the current item in a single-valued menu. *set\_menu\_mark(m,n)* sets the menu's mark string to *n*.

*menu\_mark(m)* returns a pointer to the menu's mark string.

*scale\_menu(m,rp,cp)* returns the minimum window size necessary for the given menu. *rp* and *cp* are pointers to the locations used to return the number of rows and columns for the menu.

*set\_menu\_win(m,w)* sets window *w* as the window of menu *m*.

*menu\_win(m)* returns a pointer to the menu's window.

*set\_menu\_sub(m,w)* sets window *w* as the subwindow of menu *m*.

*menu\_sub(m)* returns a pointer to the menu's subwindow.

*set\_menu\_fore(m,c)* sets the menu's foreground attribute—the display attribute for the current item (if selectable) on single-valued menus and for selected items on multi-valued menus. This display attribute is a **curses** visual attribute. By default, this attribute is **A\_STANDOUT**.

*menu\_fore(m)* returns the menu foreground attribute.

*set\_menu\_back(m,c)* sets the menu's background attribute—the display attribute for unselected, yet selectable, items. This display attribute is a **curses** visual attribute. By default, this attribute is **A\_NORMAL**.

*menu\_back(m)* returns the menu background attribute.

*set\_menu\_grey(m,c)* sets the menu's grey attribute—the display attribute for nonselectable items in multi-valued menus. This display attribute is a **curses** visual attribute. By default, this attribute is **A\_UNDERLINE**.

*menu\_grey(m)* returns the menu's grey attribute.

The pad character is the character that fills the space between a menu item's name and description, if any. *set\_menu\_pad(m,c)* sets the pad character for menu *m* to *c*.

*menu\_pad(m)* returns the menu's pad character.

*post\_menu(m)* writes the menu in the menu's subwindow.

*unpost\_menu(m)* erases the menu from its associated subwindow.



The workhorse of the menu subsystem, *menu\_driver(m,c)* checks if the character *c* is a menu request or data. If it is a request, the menu driver executes the request and reports the result. If it is data (a printable ASCII character), it enters the data into the current position in the current field. If the character is not recognized, the menu driver assumes it is an application-defined command and returns *E\_UNKNOWN\_COMMAND*.

The following *set\_* functions enable you to establish application routines to be executed automatically at initialization and termination points in your form application. You need not specify any application-defined initialization or termination routines at all, but they may be helpful for displaying messages or page numbers and other chores.

*set\_item\_init(m,f)* sets the application-defined function *f* to be called when the menu is posted and just after the current item changes.

*item\_init(m)* returns a pointer to the item initialization routine, if any, called when the menu is posted and just after the current item changes.

*set\_item\_term(m,f)* sets function *f* to be called when the menu is unposted and just before the current item changes.

*item\_term(m)* returns a pointer to the termination function, if any, called when the menu is unposted and just before the current item changes.

*set\_menu\_init(m,f)* sets the application-defined function *f* to be called when the menu is posted and just after the top row changes on a posted menu.

*menu\_init(m)* returns a pointer to the menu's initialization routine, if any, called when the menu is posted and just after the top row changes on a posted menu.

*set\_menu\_term(m,f)* sets the application-defined function *f* to be called when the menu is unposted and just before the top row changes on a posted menu.

*menu\_term(m)* returns a pointer to the menu's termination routine, if any, called when the menu is unposted and just before the top row changes on a posted menu.

The current item is the item where the cursor is currently positioned. *set\_current\_item(m,i)* sets the current menu item to the given item.

*current\_item(m)* returns a pointer to the current item.



*item\_index(i)* returns the index to the given item in the item pointer array.

*set\_top\_row(m,c)* sets the top of the menu to the named row. The left-most item on the new top row becomes the current item.

*top\_row(m)* returns the number of the menu row currently displayed at the top of the given menu.

*pos\_menu\_cursor(m)* moves the menu window's cursor to the correct position to resume menu processing.

Every menu has a pattern buffer to match entered data with menu items. *set\_menu\_pattern(m,p)* sets the pattern buffer to the given pattern and tries to find the first item that matches the pattern. If it does, the matching item becomes the current item. If not, the current item does not change.

*menu\_pattern(m)* returns the string in the pattern buffer of the given menu.

Every menu has an associated user pointer that you can use to store pertinent information.

*set\_menu\_userptr(m,n)* sets the menu's user pointer.

*menu\_userptr(m)* returns the menu's user pointer.

*set\_menu\_opts(m,o)* turns on the named options for the menu and turns off all its remaining options. Options are boolean values. Menu options are O\_ONEVALUE, O\_SHOWDESC, O\_ROWMAJOR, O\_IGNORECASE, and O\_SHOWMATCH.

*menu\_opts(m)* returns the menu's option setting.

*menu\_opts\_on(m,opts)* turns on the named options for the menu.

*menu\_opts\_off(m,opts)* turns off the named options for the menu.

## See Also

---

curses(S), field(S), fieldtype(S), form(S), item(S), panel(S), tam(S)

## Diagnostics

---

The following values are returned by one or more routines that return an integer.

**MENU (S)****E\_OK**

routine returned normally

**E\_SYSTEM\_ERROR**

system error

**E\_BAD\_ARGUMENT**

an incorrect argument was passed to the routine

**E\_POSTED**

menu is already posted

**E\_CONNECTED**

one or more items are connected to another menu

**E\_BAD\_STATE**

routine called from an inappropriate routine

**E\_NO\_ROOM**

menu does not fit within its subwindow

**E\_NOT\_POSTED**

menu has not yet been posted

**E\_UNKNOWN\_COMMAND**

unrecognizable request was given to the driver

**E\_NO\_MATCH**

no match occurred

**E\_NOT\_SELECTABLE**

item cannot be selected

**E\_NOT\_CONNECTED**

no items are associated with the menu

**E\_REQUEST\_DENIED**

menu driver could not process the request

## mkdir

---

make a directory

### Syntax

---

```
#include <sys/types.h>
#include <sys/stat.h>
```

```
int mkdir (path, mode)
char *path;
int mode;
```

### Description

---

The routine *mkdir* creates a new directory with the name *path*. The argument *mode* specifies the initial mode of the new directory. The protection bits of the argument *mode* are modified by the process file mode creation mask (see *umask(S)*). The value of the argument *mode* should be the logical OR of the values of the desired permissions:

| <i>Name</i> | <i>Description</i>                    |
|-------------|---------------------------------------|
| S_IREAD     | Read by owner                         |
| S_IWRITE    | Write by owner                        |
| S_IEXEC     | Execute (search) by owner             |
| S_IRGRP     | Read by group                         |
| S_IWGRP     | Write by group                        |
| S_IXGRP     | Execute (search) by group             |
| S_IROTH     | Read by others (that is, anyone else) |
| S_IWOTH     | Write by others                       |
| S_IXOTH     | Execute (search) by others            |

The directory's owner ID is set to the process's effective user ID. The directory's group ID is set to the process's effective group ID. The newly created directory is empty with the possible exception of entries for "." and "..". *mkdir* will fail and no directory will be created if one or more of the following is true:



|             |                                                                                                                                                       |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| [ENOTDIR]   | A component of the path prefix is not a directory.                                                                                                    |
| [ENOENT]    | A component of the path prefix does not exist.                                                                                                        |
| [ENOLINK]   | <i>path</i> points to a remote machine and the link to that machine is no longer active.                                                              |
| [EMULTIHOP] | Components of <i>path</i> require hopping to multiple remote machines.                                                                                |
| [EACCES]    | Either a component of the path prefix denies search permission, or write permission is denied on the parent directory of the directory to be created. |
| [ENOENT]    | The path is longer than the maximum allowed.                                                                                                          |
| [EEXIST]    | The named file already exists.                                                                                                                        |
| [EROFS]     | The path prefix resides on a read-only file system.                                                                                                   |
| [EFAULT]    | <i>path</i> points outside the allocated address space of the process.                                                                                |
| [EMLINK]    | The maximum number of links to the parent directory would be exceeded.                                                                                |
| [EIO]       | An I/O error has occurred while accessing the file system.                                                                                            |

## Diagnostics

---

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## Standards Conformance

---

*mkdir* is conformant with:

The X/Open Portability Guide II of January 1987;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;  
and NIST FIPS 151-1.

# mkfifo

---

make a FIFO special file

## Syntax

---

```
#include <sys/types.h>
#include <sys/stat.h>
```

```
int mkfifo (path, mode)
char *path;
mode_t mode;
```

## Description

---

The *mkfifo()* routine creates a new FIFO special file named by the pathname specified by *path*. The permissions of the new FIFO are initialized from *mode*. The file permission bits of the *mode* argument are modified by the process's file creation mask (see *umask(S)*). When bits in *mode* other than the file permission bits are set, the effect is implementation-defined.

The FIFO's owner ID is set to the process's effective user ID. The FIFO's group ID is set to the group ID of the directory in which the FIFO is being created or to the process's effective group ID.

Upon successful completion, the *mkfifo()* function marks the *st\_atime*, *st\_ctime*, and *st\_mtime* fields of the file for update. Also, the *st\_ctime* and *st\_mtime* fields of the directory that contains the new entry are marked for update.

Upon successful completion a value of zero is returned. Otherwise, a value of -1 is returned, no FIFO is created, and *errno* is set to indicate the error.

## See Also

---

*chmod(S)*, *exec(S)*, *pipe(S)*, *stat(S)*, *umask(S)*

## Diagnostics

---

If any of the following conditions occur, the *mkfifo()* function returns -1 and sets *errno* to the corresponding value:

|          |                                                          |
|----------|----------------------------------------------------------|
| [EACCES] | A component of the path prefix denies search permission. |
|----------|----------------------------------------------------------|

- [EEXIST]           The named file already exists.
- [ENAMETOOLONG]    The length of the *path* string exceeds {PATH\_MAX}, or a pathname component is longer than {NAME\_MAX} while {\_POSIX\_NO\_TRUNC} is in effect.
- [ENOENT]           A component of the path prefix does not exist or the *path* argument points to an empty string.
- [ENOSPC]           The directory that would contain the new file cannot be extended or the file system is out of file allocation resources.
- [ENOTDIR]          A component of the path prefix is not a directory.
- [EROFS]            The named file resides on a read-only file system.

## Standards Conformance

---

*mkfifo* is conformant with:

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.



## mknod

make a directory or a special or ordinary file or a FIFO

### Syntax

```
#include <sys/types.h>
#include <sys/stat.h>
int mknod (path, mode, dev)
char *path;
int mode, dev;
```

### Description

The *mknod* system call creates a new file named by the path name pointed to by *path*. The mode of the new file is initialized from *mode*. Where the value of *mode* is interpreted as follows:

|         |                                                                                                                                                                                                                                                               |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0170000 | file type; one of the following:<br>0010000 fifo special<br>0020000 character special<br>0040000 directory<br>0060000 block special<br>0100000 or 0000000 ordinary file                                                                                       |
| 0004000 | set user ID on execution                                                                                                                                                                                                                                      |
| 00020#0 | set group ID on execution if # is 7, 5, 3, or 1<br>enable mandatory file/record locking if # is 6, 4, 2, or 0                                                                                                                                                 |
| 0001000 | save text image after execution                                                                                                                                                                                                                               |
| 0000777 | access permissions; constructed from the following:<br>0000400 read by owner<br>0000200 write by owner<br>0000100 execute (search on directory) by owner<br>0000070 read, write, execute (search) by group<br>0000007 read, write, execute (search) by others |

Symbolic constants defining the value of the argument *mode* are in the *<sys/stat.h>* header file and should be used to construct *mode*. The value of the argument *mode* should be the logical OR of the values of the desired permissions:

| <i>Name</i> | <i>Description</i>                    |
|-------------|---------------------------------------|
| S_IFMT      | file type; one of the following:      |
| S_IFIFO     | FIFO-special                          |
| S_IFCHR     | character-special                     |
| S_IFDIR     | directory node                        |
| S_IFBLK     | block-special                         |
| S_IFREG     | ordinary-file                         |
| S_ISUID     | set user-ID on execution              |
| S_ISGID     | set group-ID on execution             |
| S_ISVTX     | (reserved)                            |
| S_ENFMT     | record-locking enforced               |
| S_IRUSR     | read by owner                         |
| S_IWUSR     | write by owner                        |
| S_IXUSR     | execute (search) by owner             |
| S_IRGRP     | read by group                         |
| S_IWGRP     | write by group                        |
| S_IXGRP     | execute (search) by group             |
| S_IROTH     | read by others (that is, anyone else) |
| S_IWOTH     | write by others                       |
| S_IXOTH     | execute (search) by others            |

The owner ID of the file is set to the effective user ID of the process. The group ID of the file is set to the effective group ID of the process.

Values of *mode* other than those above are undefined and should not be used. The low-order 9 bits of *mode* are modified by the process's file mode creation mask: all bits set in the process's file mode creation mask are cleared (see *umask(S)*). If *mode* indicates a block or character special file, *dev* is a configuration-dependent specification of a character or block I/O device. If *mode* does not indicate a block special or character special device, *dev* is ignored.

The *mknod* routine may be invoked only by the super-user for file types other than FIFO special.

The *mknod* routine will fail and the new file will not be created if one or more of the following is true:

|             |                                                                                          |
|-------------|------------------------------------------------------------------------------------------|
| [EPERM]     | The effective user ID of the process is not super-user.                                  |
| [ENOTDIR]   | A component of the path prefix is not a directory.                                       |
| [ENOENT]    | A component of the path prefix does not exist.                                           |
| [EROFS]     | The directory in which the file is to be created is located on a read-only file system.  |
| [EEXIST]    | The named file exists.                                                                   |
| [EFAULT]    | <i>path</i> points outside the allocated address space of the process.                   |
| [ENOSPC]    | No space is available.                                                                   |
| [EINTR]     | A signal was caught during the <i>mknod</i> system call.                                 |
| [ENOLINK]   | <i>path</i> points to a remote machine and the link to that machine is no longer active. |
| [EMULTIHOP] | Components of <i>path</i> require hopping to multiple remote machines.                   |

## See Also

---

chmod(S), exec(S), umask(S), fs(F), mkdir(C)

## Diagnostics

---

Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## Warning

---

If **mknod** is used to create a device in a remote directory (Remote File Sharing), the major and minor device numbers are interpreted by the server.



## **Standards Conformance**

---

*mknod* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## mktemp

---

make a unique file name

### Syntax

---

```
char *mktemp (template)
char *template;
```

### Description

---

The *mktemp* function replaces the contents of the string pointed to by *template* by a unique file name, and returns the address of *template*. The string in *template* should look like a file name with six trailing Xs; *mktemp* will replace the Xs with a letter and the current process ID. The letter will be chosen so that the resulting name does not duplicate an existing file.

### See Also

---

getpid(S), tmpfile(S), tmpnam(S)

### Diagnostic

---

The *mktemp* function will assign to *template* the NULL string if it cannot create a unique name.

### Notes

---

If called more than 17,576 times in a single process, this function will start recycling previously used names.

### Standards Conformance

---

*mktemp* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## mktime

---

converts local time to calendar time

### Syntax

---

```
#include <time.h>
time_t mktime(timeptr);
struct tm *timeptr; /* Local time
```

### Description

---

The *mktime* function converts the local time into a calendar value. The *timeptr* argument points to a structure that contains the local time. The structure is described in the reference page for *asctime*. The converted time has the same encoding as the values returned by the *time* function. The original values of the *tm\_wday* and *tm\_yday* components of the *timeptr* structure are ignored, and the original values of the other components are not restricted to their normal ranges.

If successful, *mktime* sets the values of *tm\_wday* and *tm\_yday* appropriately, and sets the other components to represent the specified calendar time, but with their values forced to the normal ranges; the final value of *tm\_mday* is not set until *tm\_mon* and *tm\_year* are determined.

### Notes

---

Note that the *gmtime*, *mktime*, and *localtime* functions use a single statically allocated buffer for the conversion. Each call to one of these routines destroys the result of the previous call.

### Return Value

---

The *mktime* function returns the specified calendar time encoded as a value of type *time\_t*. If the calendar time cannot be represented, the function returns the value -1 (*time\_t*).

### See Also

---

*asctime(S)*, *gmtime(S)*, *localtime(S)*, *time(S)*



## Example

---

```
#include <time.h>
#include <stdio.h>
struct tm when;
time_t now;
time_t result;
int days;
main()
{
 printf("How many days to look ahead: ");
 scanf("%d", &days);

 time(&now);
 when = *localtime(&now);
 when.tm_mday = when.tm_mday + days;
 if ((result = mktime(&when)) != (time_t)-1)
 printf("\n%d days from now the time will be %s",
 days, asctime(&when));
 else
 perror("mktime failed");
}
```

The example above takes a number of days as input and returns the future time and date on the specified number of days ahead.

## Standards Conformance

---

*mktime* is conformant with:

ANSI X3.159-198X C Language Draft Standard, May 13, 1988.

## monitor

---

prepare execution profile

### Syntax

---

```
#include <mon.h>
```

```
void monitor (lowpc, highpc, buffer, bufsize, nfunc)
int (*lowpc)(), (*highpc)();
WORD *buffer;
int bufsize, nfunc;
```

### Description

---

An executable program created by **cc -p** automatically includes calls for *monitor* with default parameters; *monitor* need not be called explicitly except to gain fine control over profiling.

The *monitor* function is an interface to *profil*(S). *lowpc* and *highpc* are the addresses of two functions; *buffer* is the address of a user-supplied array of *bufsize* WORDs (defined in the *<mon.h>* header file). *monitor* records a histogram of two items: periodically sampled values of the program counter and counts of calls to certain functions. This histogram is recorded in the buffer. The lowest address sampled is that of *lowpc* and the highest is just below *highpc*. *lowpc* may not equal 0 for this use of *monitor*. At most *nfunc*, call counts can be kept; only calls of functions compiled with the profiling option **-p** of *cc*(CP) are recorded.

For the results to be significant, especially where there are small, heavily used routines, it is suggested that the buffer be no more than a few times smaller than the range of locations sampled.

To profile the entire program, it is sufficient to use

```
extern etext;
...
monitor ((int (*)())2, &etext, buf, bufsize, nfunc);
```

*etext* lies just above all the program text; see *end*(S).

To stop execution monitoring and write the results, use

```
monitor ((int (*)())0, 0, 0, 0, 0);
```

The *prof*(CP) command can then be used to examine the results.

The name of the file written by *monitor* is controlled by the environment variable `PROFDIR`. If `PROFDIR` does not exist, "`mon.out`" is created in the current directory. If `PROFDIR` exists but has no value, *monitor* does not do any profiling and creates no output file. Otherwise, the value of `PROFDIR` is used as the name of the directory in which to create the output file. If `PROFDIR` is *dirname*, then the file written is "`dirname/pid.mon.out`" where *pid* is the program's process ID. (When *monitor* is called automatically by compiling via `cc -p`, the file created is "`dirname/pid.progname`" where *progname* is the name of the program.)

## Files

---

`mon.out`

## See Also

---

`cc(CP)`, `prof(CP)`, `profil(S)`, `end(S)`

## Notes

---

The "`dirname/pid.mon.out`" form does not work; the "`dirname/pid.progname`" form (automatically called via `cc -p`) does work.

## Standards Conformance

---

*monitor* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



# mount

---

mount a file system

## Syntax

---

```
#include <sys/types.h>
#include <sys/mount.h>
```

```
int mount (spec, dir, mflag, fstyp, dataptr, datalen)
char *spec, *dir;
int mflag, fstyp;
char *dataptr;
int datalen;
```

## Description

---

*mount* requests that a removable file system contained on the block special file identified by *spec* be mounted on the directory identified by *dir*. *spec* and *dir* are pointers to path names. *fstyp* is the file system type number. The *sysfs*(S) system call can be used to determine the file system type number. Note that if both the MS\_DATA and MS\_FSS flag bits of *mflag* are off, the file system type will default to the root file system type. Only if either flag is on will *fstyp* be used to indicate the file system type.

If the MS\_DATA flag is set in *mflag*, the system expects the *dataptr* and *datalen* arguments to be present. Together they describe a block of file-system specific data at address *dataptr* of length *datalen*. This is interpreted by file-system specific code within the operating system and its format depends upon the file system type. A particular file system type may not require this data, in which case *dataptr* and *datalen* should both be zero. Note that MS\_FSS is obsolete and will be ignored if MS\_DATA is also set, but if MS\_FSS is set and MS\_DATA is not, *dataptr* and *datalen* are both assumed to be zero.

Upon successful completion, references to the file *dir* will refer to the root directory on the mounted file system.

The low-order bit of *mflag* is used to control write permission on the mounted file system; if 1, writing is forbidden, otherwise writing is permitted according to individual file accessibility.

*mount* may be invoked only by the super-user. It is intended for use only by the *mount*(ADM) utility.

*mount* will fail if one or more of the following is true:

|             |                                                                                                          |
|-------------|----------------------------------------------------------------------------------------------------------|
| [EPERM]     | The effective user ID is not super-user.                                                                 |
| [ENOENT]    | Any of the named files does not exist.                                                                   |
| [ENOTDIR]   | A component of a path prefix is not a directory.                                                         |
| [EREMOTE]   | <i>spec</i> is remote and cannot be mounted.                                                             |
| [ENOLINK]   | <i>path</i> points to a remote machine and the link to that machine is no longer active.                 |
| [EMULTIHOP] | Components of <i>path</i> require hopping to multiple remote machines.                                   |
| [ENOTBLK]   | <i>spec</i> is not a block special device.                                                               |
| [ENXIO]     | The device associated with <i>spec</i> does not exist.                                                   |
| [ENOTDIR]   | <i>dir</i> is not a directory.                                                                           |
| [EFAULT]    | <i>spec</i> or <i>dir</i> points outside the allocated address space of the process.                     |
| [EBUSY]     | <i>dir</i> is currently mounted on, is someone's current working directory, or is otherwise busy.        |
| [EBUSY]     | The device associated with <i>spec</i> is currently mounted.                                             |
| [EBUSY]     | There are no more mount table entries.                                                                   |
| [EROFS]     | <i>spec</i> is write-protected and <i>mflag</i> requests write permission.                               |
| [ENOSPC]    | The file system state in the super-block is not FsOKAY and <i>mflag</i> requests write permission.       |
| [EINVAL]    | The super-block has an invalid magic number or the <i>fstyp</i> is invalid or <i>mflag</i> is not valid. |

## See Also

---

sysfs(S), umount(S), mount(ADM), fs(F)

## Diagnostics

---

Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## **Standards Conformance**

---

*mount* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



# msgctl

## message control operations

---

### Syntax

---

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgctl (msqid, cmd, buf)
int msqid, cmd;
msqid_ds *buf;
```

### Description

---

The *msgctl* system call provides a variety of message control operations as specified by *cmd*. The following *cmds* are available:

**IPC\_STAT** Place the current value of each member of the data structure associated with *msqid* into the structure pointed to by *buf*. The contents of this structure are defined in *intro*(S).  
{READ}

**IPC\_SET** Set the value of the following members of the data structure associated with *msqid* to the corresponding value found in the structure pointed to by *buf*:

- msg\_perm.uid
- msg\_perm.gid
- msg\_perm.mode /\* only low 9 bits \*/
- msg\_qbytes

This *cmd* can only be executed by a process that has an effective user ID equal to either that of super-user, or to the value of **msg\_perm.cuid** or **msg\_perm.uid** in the data structure associated with *msqid*. Only super-user can raise the value of **msg\_qbytes**.

**IPC\_RMID** Remove the message queue identifier specified by *msqid* from the system and destroy the message queue and data structure associated with it. This *cmd* can only be executed by a process that has an effective user ID equal to either that of super-user, or to the value of **msg\_perm.cuid** or **msg\_perm.uid** in the data structure associated with *msqid*.

The *msgctl* system call will fail if one or more of the following is true:

- |          |                                                                                                                                                                                                                                                                              |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [EINVAL] | The <i>msqid</i> argument is not a valid message queue identifier.                                                                                                                                                                                                           |
| [EINVAL] | The <i>cmd</i> argument is not a valid command.                                                                                                                                                                                                                              |
| [EACCES] | The <i>cmd</i> argument is equal to <b>IPC_STAT</b> and { <b>READ</b> } operation permission is denied to the calling process (see <i>intro</i> (S)).                                                                                                                        |
| [EPERM]  | The <i>cmd</i> argument is equal to <b>IPC_RMID</b> or <b>IPC_SET</b> . The effective user ID of the calling process is not equal to that of super-user, or to the value of <b>msg_perm.cuid</b> or <b>msg_perm.uid</b> in the data structure associated with <i>msqid</i> . |
| [EPERM]  | The <i>cmd</i> argument is equal to <b>IPC_SET</b> , an attempt is being made to increase to the value of <b>msg_qbytes</b> , and the effective user ID of the calling process is not equal to that of super-user.                                                           |
| [EFAULT] | The <i>buf</i> argument points to an illegal address.                                                                                                                                                                                                                        |

## See Also

---

*intro*(S), *msgget*(S), *msgop*(S)

## Diagnostics

---

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## Standards Conformance

---

*msgctl* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## msgget

---

get message queue

### Syntax

---

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
```

```
int msgget (key, msgflg)
key_t key;
int msgflg;
```

### Description

---

The *msgget* system call returns the message queue identifier associated with *key*.

A message queue identifier and associated message queue and data structure (see *intro*(S)) are created for *key* if one of the following is true:

The *key* argument is equal to **IPC\_PRIVATE**.

The *key* argument does not already have a message queue identifier associated with it, and (*msgflg* & **IPC\_CREAT**) is “true”.

Upon creation, the data structure associated with the new message queue identifier is initialized as follows:

**msg\_perm.cuid**, **msg\_perm.uid**, **msg\_perm.cgid**, and **msg\_perm.gid** are set equal to the effective user ID and effective group ID, respectively, of the calling process.

The low-order 9 bits of **msg\_perm.mode** are set equal to the low-order 9 bits of *msgflg*.

**msg\_qnum**, **msg\_lspid**, **msg\_lrpid**, **msg\_stime**, and **msg\_rtime** are set equal to 0.

**msg\_ctime** is set equal to the current time.

**msg\_qbytes** is set equal to the system limit.



The *msgget* system call will fail if one or more of the following is true:

- |          |                                                                                                                                                                                |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [EACCES] | A message queue identifier exists for <i>key</i> , but operation permission (see <i>intro</i> (S)) as specified by the low-order 9 bits of <i>msgflg</i> would not be granted. |
| [ENOENT] | A message queue identifier does not exist for <i>key</i> and ( <i>msgflg</i> & <i>IPC_CREAT</i> ) is "false".                                                                  |
| [ENOSPC] | A message queue identifier is to be created but the system-imposed limit on the maximum number of allowed message queue identifiers system wide would be exceeded.             |
| [EEXIST] | A message queue identifier exists for <i>key</i> but [( <i>msgflg</i> & <i>IPC_CREAT</i> ) & ( <i>msgflg</i> & <i>IPC_EXCL</i> )] is "true".                                   |

## See Also

---

*intro*(S), *msgctl*(S), *msgop*(S)

## Diagnostics

---

Upon successful completion, a non-negative integer, namely a message queue identifier, is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## Standards Conformance

---

*msgget* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## msgop: msgsnd, msgrcv

---

### message operations

### Syntax

---

```
#include <sys/types.h>
```

```
#include <sys/ipc.h>
```

```
#include <sys/msg.h>
```

```
int msgsnd (msqid, msgp, msgsz, msgflg)
```

```
int msqid;
```

```
struct msgbuf *msgp;
```

```
int msgsz, msgflg;
```

```
int msgrcv (msqid, msgp, msgsz, msgtyp, msgflg)
```

```
int msqid;
```

```
struct msgbuf *msgp;
```

```
int msgsz;
```

```
long msgtyp;
```

```
int msgflg;
```

### Description

---

The *msgsnd* system call is used to send a message to the queue associated with the message queue identifier specified by *msqid*. {WRITE} *msgp* points to a structure containing the message. This structure is composed of the following members:

|      |          |                    |
|------|----------|--------------------|
| long | msgtyp;  | /* message type */ |
| char | mtext[]; | /* message text */ |

The *msgtyp* integer is positive and can be used by the receiving process for message selection (see *msgrcv* below). The array *mtext* is any text of length *msgsz* bytes. The *msgsz* argument can range from 0 to a system-imposed maximum.

*msgflg* specifies the action to be taken if one or more of the following is true:

The number of bytes already on the queue is equal to **msg\_qbytes** (see *intro*(S)).

The total number of messages on all queues system-wide is equal to the system-imposed limit.

These actions are as follows:

If (*msgflg* & *IPC\_NOWAIT*) is "true", the message will not be sent and the calling process will return immediately.

If (*msgflg* & *IPC\_NOWAIT*) is "false", the calling process will suspend execution until one of the following occurs:

The condition responsible for the suspension no longer exists, in which case the message is sent.

The *msqid* argument is removed from the system (see *msgctl*(S)). When this occurs, *errno* is set equal to *EIDRM*, and a value of -1 is returned.

The calling process receives a signal that is to be caught. In this case the message is not sent and the calling process resumes execution in the manner prescribed in *signal*(S).

*msgsnd* will fail and no message will be sent if one or more of the following is true:

- [EINVAL]        *msqid* is not a valid message queue identifier.
- [EACCES]        Operation permission is denied to the calling process (see *intro*(S)).
- [EINVAL]        *mtype* is less than 1.
- [EAGAIN]        The message cannot be sent for one of the reasons cited above and (*msgflg* & *IPC\_NOWAIT*) is "true".
- [EINVAL]        *msgsz* is less than zero or greater than the system-imposed limit.
- [EFAULT]        *msgp* points to an illegal address.

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid* (see *intro*(S)).

*msg\_qnum* is incremented by 1.

*msg\_lspid* is set equal to the process ID of the calling process.

*msg\_stime* is set equal to the current time.

*msgrcv* reads a message from the queue associated with the message queue identifier specified by *msqid* and places it in the structure pointed to by *msgp*. {READ} This structure is composed of the following members:



```

long mtype; /* message type */
char mtext[]; /* message text */

```

*mtype* is the received message's type as specified by the sending process. *mtext* is the text of the message. *msgsz* specifies the size in bytes of *mtext*. The received message is truncated to *msgsz* bytes if it is larger than *msgsz* and (*msgflg* & **MSG\_NOERROR**) is "true". The truncated part of the message is lost and no indication of the truncation is given to the calling process.

*msgtyp* specifies the type of message requested as follows:

If *msgtyp* is equal to 0, the first message on the queue is received.

If *msgtyp* is greater than 0, the first message of type *msgtyp* is received.

If *msgtyp* is less than 0, the first message of the lowest type that is less than or equal to the absolute value of *msgtyp* is received.

*msgflg* specifies the action to be taken if a message of the desired type is not on the queue. These are as follows:

If (*msgflg* & **IPC\_NOWAIT**) is "true", the calling process will return immediately with a return value of -1 and *errno* set to ENOMSG.

If (*msgflg* & **IPC\_NOWAIT**) is "false", the calling process will suspend execution until one of the following occurs:

A message of the desired type is placed on the queue.

*msqid* is removed from the system. When this occurs, *errno* is set equal to EIDRM, and a value of -1 is returned.

The calling process receives a signal that is to be caught. In this case a message is not received and the calling process resumes execution in the manner prescribed in *signal(S)*.

*msgrcv* will fail and no message will be received if one or more of the following is true:

[EINVAL]            *msqid* is not a valid message queue identifier.

[EACCES]            Operation permission is denied to the calling process.

[EINVAL]            *msgsz* is less than 0.

[E2BIG] *mtext* is greater than *msgsz* and (*msgflg* & **MSG\_NOERROR**) is "false".

[ENOMSG] The queue does not contain a message of the desired type and (*msgtyp* & **IPC\_NOWAIT**) is "true".

[EFAULT] *msgp* points to an illegal address.

Upon successful completion, the following actions are taken with respect to the data structure associated with *msgid* (see *intro*(S)).

**msg\_qnum** is decremented by 1.

**msg\_lrp**id is set equal to the process ID of the calling process.

**msg\_rtime** is set equal to the current time.

## See Also

---

*intro*(S), *msgctl*(S), *msgget*(S), *signal*(S)

## Diagnostics

---

If *msgsnd* or *msgrcv* return due to the receipt of a signal, a value of -1 is returned to the calling process and *errno* is set to EINTR. If they return due to removal of *msgid* from the system, a value of -1 is returned and *errno* is set to EIDRM.

Upon successful completion, the return value is as follows:

*msgsnd* returns a value of 0.

*msgrcv* returns a value equal to the number of bytes actually placed into *mtext*.

Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## Standards Conformance

---

*msgrcv* and *msgsnd* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## nap

---

suspends execution for a short interval

### Syntax

---

```
long nap(period)
long period;
```

### Description

---

The current process is suspended from execution for at least the number of milliseconds specified by *period*, or until a signal is received.

### Return Value

---

On successful completion, a long integer indicating the number of milliseconds actually slept is returned. If the process received a signal while napping, the return value will be -1, and *errno* will be set to EINTR.

### See Also

---

sleep(S)

### Notes

---

This function is driven by the system clock, which in most cases has a granularity of tens of milliseconds. This function must be linked with the linker option **-lx**.



## nice

---

change priority of a process

### Syntax

---

```
int nice (incr)
int incr;
```

### Description

---

The *nice* system call adds the value of *incr* to the *nice value* of the calling process. A process's *nice value* is a non-negative number for which a more positive value results in lower CPU priority.

A maximum *nice value* of 39 and a minimum *nice value* of 0 are imposed by the system. (The default *nice value* is 20.) Requests for values above or below these limits result in the *nice value* being set to the corresponding limit.

[EPERM]            The *nice* system call will fail and not change the *nice value* if *incr* is negative or greater than 39, and the effective user ID of the calling process is not super-user.

### See Also

---

exec(S) nice(C)

### Diagnostics

---

Upon successful completion, *nice* returns the new *nice value* minus 20. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

### Standards Conformance

---

*nice* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## **nl\_ascxtime, nl\_cxtime**

---

format date and time

### **Syntax**

---

```
#include <time.h>
```

```
char *nl_cxtime(clock, format)
long *clock;
char *format;
```

```
char *nl_ascxtime(tm, format)
struct tm *tm;
char *format;
```

### **Description**

---

The routines *nl\_ascxtime* and *nl\_cxtime* are provided as an alternative interface to the *ctime*(S) routines, for programs written to the X/OPEN Portability Guide standard.

They provide the same functions as the *ctime*(S) routines *ctime* and *asctime*, except that a *strftime*(S) format string is provided to determine the layout of the resulting string.

The X/OPEN Portability Guide permits the following field descriptors only:

|   |                                       |
|---|---------------------------------------|
| n | insert a newline character            |
| t | insert a tab character                |
| m | month of year - 01 to 12              |
| d | day of month - 01 to 31               |
| y | last 2 digits of year - 00 to 99      |
| D | date as mm/dd/yy                      |
| H | hour - 00 to 23                       |
| M | minute - 00 to 59                     |
| S | second - 00 to 59                     |
| T | time as HH:MM:SS                      |
| j | day of year - 001 to 366              |
| w | day of week - 0 to 6 (Sunday = 0)     |
| a | abbreviated weekday - e.g. Sun to Sat |
| h | abbreviated month - e.g. Jan to Dec   |
| r | time in AM/PM notation                |

If *format* is the empty (zero length) string, the date and time format (D\_T\_FMT) of the currently selected *LC\_TIME* locale is used.

NL\_CXTIME (S)

NL\_CXTIME (S)

The maximum length of the resultant string is defined by the constant *NL\_TEXTMAX* in the **limits.h** file.

## See Also

---

limits(F), timtbl(M), ctime(S), nl\_init(S)

## Value Added

---

*nl\_ascxtime* and *nl\_cxtime* are extensions of AT&T System V provided by the Santa Cruz Operation.



## nl\_init

---

initializes native language support operation

### Syntax

---

```
int nl_init(lang)
char *lang;
```

### Description

---

The *nl\_init* routine is provided as an alternative interface to the *setlocale*(S) routine, for programs written to the X/OPEN Portability Guide standard.

The *nl\_init* routine initializes native language support operation for the language identified by *lang*, which is a pointer to a string containing settings of *language*, *territory* and *codeset* as defined for the LANG environment variable (see *environ*(M)).

Typically, *nl\_init* is used to bind program operation to the user's specified language requirements, for example:

```
nl_init(getenv("LANG"));
```

A call to *nl\_init* will fail if the string pointed to by *lang* does not identify a valid *language/territory/codeset* combination. In this case, operation will continue for the language identified on the last successful call.

### Return Value

---

If successful, *nl\_init* will return 0 (zero). Otherwise, -1 will be returned.

### See Also

---

*environ*(M), *locale*(M), *getenv*(S), *setlocale*(S)

## Notes

---

Calls to *nl\_init* can be used to switch operation from one supported language to another. It is not necessary to call *nl\_init* to set the initial language for an application; this is done by the automatic *setlocale* call at program startup.

The *nl\_init* routine is not as flexible an interface as that provided by *setlocale*.

## Value Added

---

*nl\_init* is an extension of AT&T System V provided by the Santa Cruz Operation.

## nl\_langinfo

---

language information

### Syntax

---

```
#include <nl_types.h>
#include <langinfo.h>

char *nl_langinfo(item)
nl_item item;
```

### Description

---

The *nl\_langinfo* routine returns a pointer to a null terminated string that contains information relevant to a particular language or cultural area identified by the last successful call to *setlocale* or *nlinit*. The constant names and values of *item* are defined in the file **langinfo.h**.

Here is an example. The following statement sets a character pointer called **day** to point to the string "Dom" if the identified language is Portuguese or "Sun" if the identified language is English.

```
day = nl_langinfo(ABDAY_1)
```

If the parameter is not a valid *item*, *nl\_langinfo* returns a pointer to an empty (null) string.

### See Also

---

langinfo(F), nl\_types(F), environ(M), nl\_init(S), setlocale(S)

### Value Added

---

*nl\_langinfo* is an extension of AT&T System V provided by the Santa Cruz Operation.



## **nl\_printf, nl\_fprintf, nl\_sprintf**

---

formats native language output

### **Syntax**

---

```
int nl_printf(format [, arg ...])
char *format;
```

```
int nl_fprintf(stream, format [, arg ...])"
FILE *stream;
char *format;
```

```
int nl_sprintf(s, format [, arg ...])
char *s, *format;
```

### **Description**

---

The functions *nl\_printf*, *nl\_fprintf* and *nl\_sprintf* provide similar functionality to the standard *printf*, *fprintf* and *sprintf* routines (see *printf*(S)), with the difference that the conversion character *%* in the format string is replaced by the sequence *%digit\$*, where *digit* is a decimal digit *n* in the range (1-{ NL\_ARGMAX }) (see *limits*(F)). Conversions are applied to the *n*th argument in the argument list, rather than to the next unused argument.

The *format* passed to these functions can contain either form of conversion specification, namely *%* or *%digit\$*, although the two forms cannot be mixed within a single *format* string. The format string should contain values for all the arguments specified in the argument list. The asterisk, *\**, cannot be used to indicate a field width in the format string.

### **See Also**

---

*limits*(F), *nl\_init*(S), *nl\_scanf*(S), *printf*(S)

### **Examples**

---

The following *nl\_printf* statement could be used to print the language independent date and time format:

```
nl_printf(format, weekday, month, day, hour, min);
```

For American usage, *format* could be a pointer to the string:

"%1\$s, %2\$s %3\$d, \$4\$d:%5\$.2d0

This would produce the following message:

Sunday, July 3, 10:02

For German usage, *format* could be a pointer to the string:

"%1\$s, %3\$d. %2\$s, \$4\$d:%5\$.2d0

This would produce the message:

Sonntag, 3. Juli, 10:02

## Value Added

---

*nl\_fprintf*, *nl\_printf* and *nl\_sprintf* are extensions of AT&T System V provided by the Santa Cruz Operation.

## **nl\_scanf, nl\_fscanf, nl\_sscanf**

---

converts formatted native language input

### **Syntax**

---

```
int nl_scanf(format [, pointer ...])
char *format;
```

```
int nl_fscanf(stream, format [, pointer ...])
FILE *stream;
char *format;
```

```
int nl_sscanf(s, format [, pointer ...])
char *s, *format;
```

### **Description**

---

The functions *nl\_scanf*, *nl\_fscanf* and *nl\_sscanf* provide similar functionality to the standard *scanf*, *fscanf* and *sscanf* routines (see *scanf(S)*), with the difference that the conversion character % in the format string is replaced by the sequence *%digit\$*, where *digit* is a decimal digit *n* in the range (1-{ NL\_ARGMAX }) (see *limits(F)*). Conversions are applied to the *n*th argument in the argument list, rather than to the next unused argument.

The *format* passed to these functions can contain either form of conversion specification, such as % or *%digit\$*, although the two forms cannot be mixed within a single *format* string.

### **See Also**

---

*limits(F)*, *nl\_init(S)*, *nl\_printf(S)*, *scanf(S)*

### **Value Added**

---

*nl\_fscanf*, *nl\_scanf* and *nl\_sscanf* are extensions of AT&T System V provided by the Santa Cruz Operation.



## **nl\_strcmp, nl\_strncmp**

---

compare native language strings

### **Syntax**

---

```
int *nl_strcmp(s1, s2)
char *s1, *s2;
```

```
int *nl_strncmp(s1, s2, n)
char *s1, *s2;
int n;
```

### **Description**

---

The routines *nl\_strcmp* and *nl\_strncmp* are provided as an alternative interface to the *strcoll*(S) routine, for programs written to the X/OPEN Portability Guide standard. They provide the same functions as the *string*(S) routines *strcmp* and *strncmp*, except that the language dependent collating information defined for the current LC\_COLLATE locale is used to rank the strings, instead of the binary values of the machine character set.

In this way, the strings are compared according to the dictionary order of data, taking into account case and accent priority, 1-to-1, 1-to-2, 2-to-1 and don't care character mappings.

### **See Also**

---

coltbl(M), collation(S), nl\_init(S), string(S)

### **Value Added**

---

*nl\_strcmp* and *nl\_strncmp* are extensions of AT&T System V provided by the Santa Cruz Operation.

## nlist

---

get entries from name list

### Syntax

---

```
#include <nlist.h>
```

```
int nlist (filename, nl)
char *filename;
struct nlist *nl;
```

### Description

---

The *nlist* function examines the name list in the executable file whose name is pointed to by *filename*, and selectively extracts a list of values and puts them in the array of *nlist* structures pointed to by *nl*. The name list *nl* consists of an array of structures containing names of variables, types, and values. The list is terminated with a null name; that is, a null string is in the name position of the structure. Each variable name is looked up in the name list of the file. If the name is found, the type and value of the name are inserted in the next two fields. The type field will be set to 0 unless the file was compiled with the *-g* option. If the name is not found, both entries are set to 0. See *a.out*(F) for a discussion of the symbol table structure.

This function is useful for examining the system name list kept in the file */unix*. In this way programs can obtain system addresses that are up to date.

### See Also

---

*a.out*(F)

### Diagnostics

---

All value entries are set to 0 if the file cannot be read or if it does not contain a valid name list.

The *nlist* function returns -1 upon error; otherwise it returns 0.

## Notes

---

The `<nlist.h>` header file is automatically included by `<a.out.h>` for compatibility. However, if the only information needed from `<a.out.h>` is for use of *nlist*, then including `<a.out.h>` is discouraged. If `<a.out.h>` is included, the line “`#undef n_name`” may need to follow it.

## Standards Conformance

---

*nlist* is conformant with:

AT&T SVID Issue 2, Select Code 307-127.



## open

---

open for reading or writing

### Syntax

---

```
#include <fcntl.h>
int open (path, oflag [, mode])
char *path;
int oflag, mode;
```

### Description

---

*path* points to a path name naming a file. The *open* system call opens a file descriptor for the named file and sets the file status flags according to the value of *oflag*. For non-STREAMS (see *intro(S)*) files, *oflag* values are constructed by OR-ing flags from the following list (only one of the first three flags below may be used):

**O\_RDONLY**

Open for reading only.

**O\_WRONLY**

Open for writing only.

**O\_RDWR** Open for reading and writing.

**O\_NDELAY**

This flag may affect subsequent reads and writes (see *read(S)* and *write(S)*).

When opening a FIFO with **O\_RDONLY** or **O\_WRONLY** set:

If **O\_NDELAY** is set:

An *open* for reading-only will return without delay. An *open* for writing-only will return an error if no process currently has the file open for reading.

If **O\_NDELAY** is clear:

An *open* for reading-only will block until a process opens the file for writing. An *open* for writing-only will block until a process opens the file for reading.

When opening a file associated with a communication line:

If `O_NDELAY` is set:

The open will return without waiting for carrier.

If `O_NDELAY` is clear:

The open will block until carrier is present.

### **O\_APPEND**

If set, the file pointer will be set to the end of the file prior to each write.

### **O\_SYNC**

When opening a regular file, this flag affects subsequent writes. If set, each *write*(S) will wait for both the file data and file status to be physically updated.

### **O\_CREAT**

If the file exists, this flag has no effect. Otherwise, the owner ID of the file is set to the effective user ID of the process; the group ID of the file is set to the effective group ID of the process; and the low-order 12 bits of the file mode are set to the value of *mode*, modified as follows [see *creat*(S)]:

All bits set in the file mode creation mask of the process are cleared (see *umask*(S)).

The "save text image after execution bit" of the mode is cleared (see *chmod*(S)).

### **O\_TRUNC**

If the file exists, its length is truncated to 0 and the mode and owner are unchanged.

### **O\_EXCL**

If `O_EXCL` and `O_CREAT` are set, *open* will fail if the file exists.

When opening a STREAMS file, *oflag* may be constructed from `O_NDELAY` or-ed with either `O_RDONLY`, `O_WRONLY` or `O_RDWR`. Other flag values are not applicable to STREAMS devices and have no effect on them. The value of `O_NDELAY` affects the operation of STREAMS drivers and certain system calls (see *read*(S), *getmsg*(S), *putmsg*(S), and *write*(S)). For drivers, the implementation of `O_NDELAY` is device-specific. Each STREAMS device driver may treat this option differently.

Certain flag values can be set following *open* as described in *fcntl*(S).

The file pointer used to mark the current position within the file is set to the beginning of the file.

The new file descriptor is set to remain open across *exec* system calls (see *fcntl*(S)).

The named file is opened unless one or more of the following is true:

- |             |                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------|
| [EACCES]    | A component of the path prefix denies search permission.                                                                          |
| [EACCES]    | <i>oflag</i> permission is denied for the named file.                                                                             |
| [EAGAIN]    | The file exists, mandatory file/record locking is set, and there are outstanding record locks on the file (see <i>chmod</i> (S)). |
| [EEXIST]    | O_CREAT and O_EXCL are set, and the named file exists.                                                                            |
| [EFAULT]    | <i>path</i> points outside the allocated address space of the process.                                                            |
| [EINTR]     | A signal was caught during the <i>open</i> system call.                                                                           |
| [EIO]       | A hangup or error occurred during a STREAMS <i>open</i> .                                                                         |
| [EISDIR]    | The named file is a directory and <i>oflag</i> is write or read/write.                                                            |
| [EMFILE]    | No file descriptors are currently open.                                                                                           |
| [EMULTIHOP] | Components of <i>path</i> require hopping to multiple remote machines.                                                            |
| [ENFILE]    | The system file table is full.                                                                                                    |
| [ENOENT]    | O_CREAT is not set and the named file does not exist.                                                                             |
| [ENOLINK]   | <i>path</i> points to a remote machine, and the link to that machine is no longer active.                                         |
| [ENOMEM]    | The system is unable to allocate a send descriptor.                                                                               |
| [ENOSPC]    | O_CREAT and O_EXCL are set, and the file system is out of inodes.                                                                 |
| [ENOSR]     | Unable to allocate a <i>stream</i> .                                                                                              |
| [ENOTDIR]   | A component of the path prefix is not a directory.                                                                                |
| [ENXIO]     | The named file is a character special or block special file, and the device associated with this special file does not exist.     |



- [ENXIO] O\_NDELAY is set, the named file is a FIFO, O\_WRONLY is set, and no process has the file open for reading.
- [ENXIO] A STREAMS module or driver open routine failed.
- [EROFS] The named file resides on a read-only file system and *oflag* is write or read/write.
- [ETXTBSY] The file is a pure procedure (shared text) file that is being executed and *oflag* is write or read/write.

## See Also

---

chmod(S), close(S), creat(S), dup(S), fcntl(S), intro(S), lseek(S), read(S), getmsg(S), putmsg(S), umask(S), write(S)

## Diagnostics

---

Upon successful completion, the file descriptor is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## Standards Conformance

---

*open* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

## opensem

---

opens a semaphore

### Syntax

---

```
int opensem(sem_name)
char *sem_name;

sem_num = opensem(sem_name);
```

### Description

---

*opensem* opens a semaphore named by *sem\_name* and returns the unique semaphore identification number *sem\_num* used by *waitsem* and *sigsem*. *creatsem* should always be called to initialize the semaphore before the first attempt to open it.

### System Compatibility

---

*opensem* can only be used to open semaphores created under UNIX version 3.0, not for UNIX System V semaphores.

### See Also

---

*creatsem*(S), *sigsem*(S), *waitsem*(S)

### Diagnostics

---

*opensem* returns a value of -1 if an error occurs. If the semaphore named does not exist, *errno* is set to ENOENT. If the file specified is not a semaphore file (i.e., a file previously created by a process using a call to *creatsem*), *errno* is set to ENOTNAM. If the semaphore has become invalid due to inappropriate use, *errno* is set to ENAVAIL.

### Notes

---

This feature is a XENIX specific enhancement which may not be present in all UNIX implementations. This function must be linked with the linker option **-lx**.

**Warning**

---

It is not advisable to open the same semaphore more than once. Although it is possible to do this, it may result in a serious deadlock.



## **paccess**

---

used in conjunction with *ptrace* for tracing a child process

### **Syntax**

---

```
#include <sys/paccess.h>
```

```
paccess(pid,cmd,offset,count,ptr)
int pid, cmd,offset,count;
char *ptr;
```

### **Description**

---

*paccess*(S) provides an extended interface for accessing the address space, register save areas and local descriptor table of a child process that is being traced using *ptrace*(CP). *paccess* also provides information about the u-area layout so that applications can determine this at runtime. This allows the developer to avoid hard coding kernel dependencies into the application.

The primary use of *paccess* is in the implementation of debuggers such as *adb*(CP) and *sdb*(CP).

### **Parameters**

---

Except for *P RUOFFS*, all *paccess* commands transfer information between the calling process and a child process designated by *pid*. The parent and child processes must coordinate tracing using the facilities described by *ptrace*. Each command selects a particular class of data such as data address space or floating point register save area, and is used by the parent process to read or write the child process's context.

*offset* is a byte granularity logical offset from the base of the particular data area (determined by *paccess* relative to *cmd*).

*count* is the size in bytes of the requested transfer. *count* may be sized down by *paccess* if *offset* plus *count* would exceed the size of the particular data area. *count* may not exceed *MAXIPCDATA*.

*ptr* designates a buffer in the caller's address space.

## Note

---

*paccess* will transfer a maximum of *count* bytes between a buffer designated by *ptr* and a point *offset* bytes into a region of the child process's context designated by *cmd*.

*P\_RUOFFS* is an exception as indicated below.

## Commands

---

The first four commands are used to read or write data in the data, text or stack regions of a process:

- |          |                                                                                                                                                                                                                                                                                                                                                                                       |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| P_RDUSER | Reads a child process's D address space. <i>offset</i> is a 32 byte virtual address in the child's address space.                                                                                                                                                                                                                                                                     |
| P_RIUSER | Reads a child process's I address space. <i>offset</i> is a 32 byte virtual address in the child's address space.                                                                                                                                                                                                                                                                     |
| P_WDUSER | Writes to a child process's D address space. <i>offset</i> is a 32 byte virtual address in the child's address space.                                                                                                                                                                                                                                                                 |
| P_WIUSER | Writes to a child process's I address space. <i>offset</i> is a 32 byte virtual address in the child's address space.                                                                                                                                                                                                                                                                 |
| P_RUREGS | Reads the child process's register save area in the uarea. <i>offset</i> is a logical offset from the base of the register save area in the uarea. This may be used to read a particular register or a subset of the entire save area. Note that the ordering of registers in the save area is highly machine dependent. The offsets of the registers are defined in <sys/reg.h>.     |
| P_WUREGS | Writes to the child process's register save area in the uarea. <i>offset</i> is a logical offset from the base of the register save area in the uarea. This may be used to read a particular register or a subset of the entire save area. Note that the ordering of registers in the save area is highly machine dependent. The offsets of the registers are defined in <sys/reg.h>. |

The kernel stack pointer [KESP] will remain unchanged after any instance of P\_WUREGS. Certain flags in the flags register [EFL] will remain unchanged after any instance of P\_WUREGS. The flags are:



PS\_T  
PS\_IE  
PS\_IOPL  
PS\_NT  
PS\_RF  
PS\_VM

See the file `<sys/tss.h>` for a definition of these flags.

**P\_RUFREGS** Reads the child process floating point register save area in the uarea. *offset* is a logical offset from the base of the register save area in the uarea. This may be used to read a particular register or a subset of the entire save area.

**P\_WUFREGS** Writes to the child process floating point register save area in the uarea. *offset* is a logical offset from the base of the register save area in the uarea. This may be used to read a particular register or a subset of the entire save area.

For the above two commands *paccess* selects the appropriate save area relative to whether the process is using an Intel 87 series or Weitek co-processor chip and whether or not the hardware is present or is being emulated in software. Note that data is not currently transformed in any way and is highly chip and/or emulator dependent.

The ordering of registers in the save area is highly machine dependent. The offsets of the registers are defined in `<sys/reg.h>`.

**P\_RULDT** Read the child processor's local descriptor table. This command has no write capability. *offset* is a logical byte offset from the base of the local descriptor table. *paccess* may have to be invoked several times to read the entire local descriptor table.

The following commands provide read and write access to the 80386 debug register save area.

**P\_RUDREGS** Reads the 386 debug register save area. *offset* is a logical offset from the base of the register save area in the uarea. This command may be used to read a particular register or a subset of the entire save area.



**P\_WUDREGS** Writes to the 386 debug register save area. *offset* is a logical offset from the base of the register save area in the uarea. This command may be used to read a particular register or a subset of the entire save area.

The 80386 debug register set may be used to implement text and data breakpoints. The layout and format of the debug register save area is highly chip dependent. The offsets of the registers are defined in `<sys/reg.h>`.

Certain fields, such as global bits, in the status register will remain unaffected by any instance of **P\_WUDREGS**. See `/usr/include/sys/debugreg.h` for the definition of any flags.

The file `/usr/include/sys/paccess.h` includes a structure template for reading and writing the debug register save area:

#### **struct debugregs**

This may be used in conjunction with `/usr/include/sys/debugreg.h` for convenient handling of bitwise operations.

**P\_RUOFFS** This command is used to obtain a list of kernel dependent uarea offsets typically used by debuggers such as *adb* and *sdb*. This allows an application developer to avoid using hard coded values and thereby gain greater independence from specific kernel versions.

*pid* and *offset* are ignored by this command.

The command returns the first *count* bytes of the offsets structure defined in *paccess.h*. That structure is defined as follows:

```
typedef long uoff;

/* 3.2 uarea offsets */

struct uoffsets {
 uoff u_info; /* version */
 uoff u_uaddr; /* kernel virtual address of uarea */
 uoff u_ar0; /* user register save area pointer */
 uoff u_fps; /* floating point save area */
 uoff u_fpemul; /* separate emulator save area */
 uoff u_fpvalid; /* if floating point save is valid */
 uoff u_weitek; /* per proc weitek flag */
 uoff u_weitek_reg /* weitek save area */
 uoff u_debugreg; /* debug register save area */
 uoff u_ldt; /* offset of ldt */
 uoff u_ldtlimit; /* size of ldt */
 uoff u_tss; /* 3.2 adb */
 uoff u_sztss; /* 3.2 adb */
 uoff u_sigreturn; /* user signal return */
};
```

## Errors

---

*paccess* will fail if one or more of the following is true:

1. *cmd* is invalid [EINVAL].
2. *pid* identifies a child that does not exist or has not executed a *ptrace* with request 0 [ESRCH].
3. *offset* is less than 0 or beyond the size of the relevant structure [EINVAL].
4. *ptr* points outside the allocated address space [EFAULT].
5. An error was encountered when attempting to access data in the child's address space [EIO].

## Return Value

---

Upon successful completion, *paccess* returns the number of bytes successfully transferred to or from the child process. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## Files

---

/usr/include/sys/paccess.h

## See Also

---

ptrace(S), access(S)

## Value Added

---

*paccess* is an extension of AT&T System V provided by the Santa Cruz Operation.



# panel

---

## PANEL library routines

### Syntax

---

**#include** <panel.h>

cc [ flags ] files -lpanel -lcurses [ libraries ]  
PANEL \*new\_panel(win)  
WINDOW \*win;

WINDOW \*panel\_window(panel)  
PANEL \*panel;

int replace\_panel(panel, window)  
PANEL \*panel;  
WINDOW \*window;

int move\_panel(panel, starty, startx)  
PANEL \*panel;  
int starty, startx;

int bottom\_panel(panel)  
PANEL \*panel;

int top\_panel(panel)  
PANEL \*panel;

void update\_panels()

int hide\_panel(panel)  
PANEL \*panel;

int panel\_hidden(panel)  
PANEL \*panel;

int show\_panel(panel)  
PANEL \*panel;

PANEL \*panel\_above(panel)  
PANEL \*panel;

PANEL \*panel\_below(panel)  
PANEL \*panel;

int \*set\_panel\_userptr(panel, ptr)  
PANEL \*panel;  
char \*ptr;

```
char *panel_userptr(panel)
PANEL *panel;
```

```
int del_panel(panel)
PANEL *panel;
```

## Description

---

Panels are rectangles of text with depth. They enable your windows to overlap without having hidden portions of underlying windows be mistakenly visible. **stdscr** lies beneath all panels. The set of currently visible panels is the **deck** of panels.

A window is associated with every panel. The panel routines enable you to create panels, fetch their associated windows, shuffle panels in the deck, and manipulate panels in other ways.

PANEL routines run on any terminal supported by **curses(S)**, the low-level Extended Terminal Interface (ETI) library. Once you compile your ETI program **#include**ing the PANEL header file **panel.h**, you should link it with the **panel** and **curses** library routines.

## Functions

---

For a complete description of each panel routine, see the *Programmer's Guide*.

*new\_panel(win)* returns a pointer to a new panel associated with *win*. The new panel is placed on top of the panel deck.

*panel\_window(panel)* returns a pointer to the window of *panel*.

*replace\_panel(panel, window)* replaces the current window of *panel* with *window*.

*move\_panel(panel, starty, startx)* moves the given panel window so that its upper-left corner is at *starty, startx*. Be sure to use this function, not **mvwin()**, to move a panel window.

*bottom\_panel(panel)* puts *panel* at the bottom of all panels. It leaves the size and contents of its associated window, and its relations to other panels, wholly intact.

*top\_panel(panel)* puts the given visible panel on top of all panels in the deck.

*void update\_panels()* refreshes the virtual screen to reflect the relations between the panels in the deck, but does not call *doupdate()* to refresh the physical screen.

*hide\_panel (panel)* removes the panel from the panel deck and thus hides it from view. The panel's internal data structure, however, is retained.

*panel\_hidden (panel)* returns a boolean value indicating whether or not the given panel has been removed from the panel deck.

*show\_panel (panel)* makes a hidden panel visible by placing it on top of the panels in the panel deck.

*panel\_above (panel)* returns a pointer to the panel just above **panel**. If the panel argument is NULL, i.e., (**panel \***) 0, it returns a pointer to the bottom panel in the deck.

*panel\_below (panel)* returns a pointer to the panel just below **panel**. If the panel argument is NULL, it returns a pointer to the top panel in the deck.

*set\_panel\_userptr (panel ,ptr)* sets the panel's user pointer.

*panel\_userptr (panel)* returns the user pointer for a given panel.

*del\_panel (panel)* deletes the panel, but not its associated window.

## See Also

---

curses(S), field(S), fieldtype(S), form(S), item(S), menu(S), tam(S)

## Diagnostics

---

Each panel routine that returns a pointer to an object returns NULL if an error occurs. Each panel routine that returns an **int** value returns OK if it executes successfully and ERR if not.



## passlen

---

determine minimum password length

### Syntax

---

```
#include <sys/types.h>
#include <sys/security.h>
#include <sys/audit.h>
#include <prot.h>
```

```
int passlen(life_dur, login_delay, alphabet_size)
time_t life_dur;
time_t login_delay;
int alphabet_size;
```

### Description

---

The *passlen* routine takes the password lifetime duration *life\_dur* for an account and the *login\_delay* between login attempts, both values in seconds, and the *alphabet\_size* of distinct characters, and returns the minimum password length according to the algorithm in the *DoD Password Management Guideline (Green Book)*. Two other parameters used by the *Guideline* are fixed, namely the size of the alphabet (26 characters) and the probability of guessing a password (1 chance in a million).

The formula is:

$$\text{min\_pass\_len} = \text{ceil} \left[ \frac{\ln \frac{\text{login\_delay} \times \text{life\_dur}}{\text{prob\_guess}}}{\ln \text{alphabet\_size}} \right]$$

### Notes

---

If the lifetime or guess parameters change in the Protected Password database for this account (or for the system default if those values are used in an account), it is a good idea to invalidate the password in case the new parameters require a longer minimum password length.

### See Also

---

passwd(C), randomword(S), exp(S), floor(S)  
*DoD Password Management Guideline (Green Book)*,  
 CSC-STD-002-85, 12 April 1985.

**Value Added**

---

*passlen* is an extension of AT&T System V provided by the Santa Cruz Operation.

# pathconf

get configurable pathname variables

## Syntax

```
#include <unistd.h>
```

```
long pathconf (path,name)
char *path;
int name;
```

```
long fpathconf (fildes,name)
int fildes, name;
```

## Description

The *pathconf()* and *fpathconf()* functions provide a method for the application to determine the current value of a configurable limit or option (*variable*) that is associated with the file or directory.

For *pathconf()*, the *path* argument points to the pathname of a file or directory. For *fpathconf()*, the *fildes* argument is an open file descriptor.

The *name* argument represents the variable to be queried relative to the file or directory. The implementation supports all of the variables listed in the following table and may support others. The variables in the following table come from *<limits.h>* or *<unistd.h>* and the symbolic constants, defined in *<unistd.h>*, that are the corresponding values used for *name*.

| Variable                  | <i>name</i> Value      | Notes |
|---------------------------|------------------------|-------|
| {LINK_MAX}                | {_PC_LINK_MAX}         | 1     |
| {MAX_CANON}               | {_PC_MAX_CANON}        | 2     |
| {MAX_INPUT}               | {_PC_MAX_INPUT}        | 2     |
| {NAME_MAX}                | {_PC_NAME_MAX}         | 3,4   |
| {PATH_MAX}                | {_PC_PATH_MAX}         | 4,5   |
| {PIPE_BUF}                | {_PC_PIPE_BUF}         | 6     |
| {_POSIX_CHOWN_RESTRICTED} | {_PC_CHOWN_RESTRICTED} | 7     |
| {_POSIX_NO_TRUNC}         | {_PC_NO_TRUNC}         | 3,4   |
| {_POSIX_VDISABLE}         | {_PC_VDISABLE}         | 2     |

Note that:

1. If *path* or *fildes* refers to a directory, the value returned applies to the directory itself.



2. The behavior is undefined if *path* or *fildes* does not refer to a terminal file.
3. If *path* or *fildes* refers to a directory, the value returned applies to the filenames within the directory.
4. The behavior is undefined if *path* or *fildes* does not refer to a directory.
5. If *path* or *fildes* refers to a directory, the value returned is the maximum length of a relative pathname when the specified directory is the working directory.
6. If *path* refers to FIFO, or *fildes* refers to a pipe or FIFO, the value returned applies to the referenced object itself. If *path* or *fildes* refers to a directory, the value returned applies to any FIFOs that exist or can be created within the directory. If *path* or *fildes* refer to any other type of file, the behavior is undefined.
7. If *path* or *fildes* refer to a directory, the value returned applies to any files defined in this standard, other than directories, that exist or can be created within this directory.

## Return Value

---

If *name* is an invalid value, the *pathconf()* and *fpathconf()* functions return -1.

If the variable corresponding to *name* has no limit for the path or file descriptor, the *pathconf()* and *fpathconf()* functions return a -1 without changing *errno*.

If the implementation needs to use *path* to determine the value of *name* and the implementation does not support the association of *name* with the file specified by *path*, or if the process did not have the appropriate privileges to query the file specified by *path*, or *path* does not exist, the *pathconf()* function returns -1.

If the implementation needs to use *fildes* to determine the value of *name* and the implementation does not support the association of *name* with the file specified by *fildes*, or if *fildes* is an invalid descriptor, the *fpathconf()* function returns -1.

Otherwise, the *pathconf()* and *fpathconf()* functions return the current variable value for the file or directory without changing *errno*. The value returned is not more restrictive than the corresponding value described to the application when it was compiled with the implementation's `<limits.h>` or `<unistd.h>`.

## Diagnostics

---

If any of the following conditions occur, the *pathconf()* and *fpathconf()* functions return -1 and set *errno* to the corresponding value:

[EINVAL]           The value of name is invalid.

For each of the following conditions, if the condition is detected, the *pathconf()* function returns a -1 and set *errno* to the following value:

[EACCES]           Search permission is denied for a component of the path prefix.

[EINVAL]           The implementaion does not support an association of the variable name with the specified file.

[ENAMETOOLONG]     The length of the *path* argument exceeds {PATH\_MAX} or a pathname component is longer than {NAME\_MAX} while {POSIX\_NO\_TRUNC} is in effect.

[ENOENT]           The named file does not exist or the *path* argument points to an empty string.

[ENOTDIR]           A component of the path prefix is not a directory.

For each of the following conditions, if the condition is detected, the *fpathconf()* function returns a -1 and set *errno* to the corresponding value:

[EBADF]            The *fildev* argument is not a valid file descriptor.

[EINVAL]           The implementation does not support an association of the variable name with the specified file.

## Standards Conformance

---

*pathconf* is conformant with:

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

## pause

---

suspend process until signal

### Syntax

---

`pause ( )`

### Description

---

The *pause* system call suspends the calling process until it receives a signal. The signal must be one that is not currently set to be ignored by the calling process.

If the signal causes termination of the calling process, *pause* will not return.

If the signal is *caught* by the calling process, and control is returned from the signal-catching function (see *signal(S)*), the calling process resumes execution from the point of suspension; with a return value of -1 from *pause* and *errno* set to EINTR.

### See Also

---

`alarm(S)`, `kill(S)`, `signal(S)`, `sigpause(S)`, `wait(S)`

### Standards Conformance

---

*pause* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.



# **perror, errno, sys\_errlist, sys\_nerr**

system error messages

## **Syntax**

```
void perror (s)
const char *s;

extern int errno;

extern char *sys_errlist[];

extern int sys_nerr;
```

## **Description**

The *perror* function produces a message on the standard error output, describing the last error encountered during a call to a system or library function. The argument string *s* is printed first, then a colon and a blank, then the message and a new-line. (However, if *s*="", the colon is not printed.) To be of most use, the argument string should include the name of the program that incurred the error. The error number is taken from the external variable *errno*, which is set when errors occur but not cleared when non-erroneous calls are made.

To simplify variant formatting of messages, the array of message strings *sys\_errlist* is provided; *errno* can be used as an index into this table to get the message string without the new-line. *sys\_nerr* is the number of messages in the table; it should be checked because new error codes may be added to the system before they are added to the table.

## **See Also**

intro(S)

## Standards Conformance

---

*errno* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;  
and NIST FIPS 151-1.

*perror* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
ANSI X3.159-198X C Language Draft Standard, May 13,  
1988;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;  
and NIST FIPS 151-1.

*sys\_errlist* and *sys\_nerr* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## pipe

---

create an interprocess channel

### Syntax

---

```
int pipe (fildes)
int fildes[2];
```

### Description

---

The *pipe* system call creates an I/O mechanism called a pipe and returns two file descriptors, *fildes*[0] and *fildes*[1]. *fildes*[0] is opened for reading and *fildes*[1] is opened for writing.

Up to 5120 bytes of data are buffered by the pipe before the writing process is blocked. A read-only file descriptor *fildes*[0] accesses the data written to *fildes*[1] on a first-in-first-out (FIFO) basis.

The *pipe* system call will fail if:

- |          |                                                          |
|----------|----------------------------------------------------------|
| [EMFILE] | The current process has reached the limit of open files. |
| [ENFILE] | The system file table is full.                           |

### See Also

---

read(S), write(S), sh(C)

### Diagnostics

---

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

### Standards Conformance

---

*pipe* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.



## plock

lock process, text, or data in memory

### Syntax

```
#include <sys/lock.h>
```

```
int plock (op)
int op;
```

### Description

The *plock* system call allows the calling process to lock its text segment (text lock), its data segment (data lock), or both its text and data segments (process lock) into memory. Locked segments are immune to all routine swapping. *plock* also allows these segments to be unlocked. The effective user ID of the calling process must be super-user to use this call. *op* specifies the following:

- PROCLOCK** - lock text and data segments into memory (process lock)
- TEXTLOCK** - lock text segment into memory (text lock)
- DATLOCK** - lock data segment into memory (data lock)
- UNLOCK** - remove locks

The *plock* system call will fail and not perform the requested operation if one or more of the following is true:

- [EPERM] The effective user ID of the calling process is not super-user.
- [EINVAL] *op* is equal to **PROCLOCK** and a process lock, a text lock, or a data lock already exists on the calling process.
- [EINVAL] *op* is equal to **TEXTLOCK** and a text lock or a process lock already exists on the calling process.
- [EINVAL] *op* is equal to **DATLOCK** and a data lock or a process lock already exists on the calling process.
- [EINVAL] *op* is equal to **UNLOCK** and no type of lock exists on the calling process.

PLOCK (S)

PLOCK (S)

[EAGAIN]

Not enough memory.

## See Also

---

exec(S), exit(S), fork(S)

## Diagnostics

---

Upon successful completion, a value of 0 is returned to the calling process. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## Standards Conformance

---

*plock* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## plot

---

### graphics interface subroutines

#### Syntax

---

**openpl** ( )

**erase** ( )

**label** (s)  
**char** \*s;

**line** (x1, y1, x2, y2)  
**int** x1, y1, x2, y2;

**circle** (x, y, r)  
**int** x, y, r;

**arc** (x, y, x0, y0, x1, y1)  
**int** x, y, x0, y0, x1, y1;

**move** (x, y)  
**int** x, y;

**cont** (x, y)  
**int** x, y;

**point** (x, y)  
**int** x, y;

**linemod** (s)  
**char** \*s;

**space** (x0, y0, x1, y1)  
**int** x0, y0, x1, y1;

**closepl** ( )

#### Description

---

These subroutines generate graphic output in a relatively device-independent manner. *space* must be used before any of these functions to declare the amount of space necessary (see *plot(F)*). *openpl* must be used before any of the others to open the device for writing. *closepl* flushes the output.



*circle* draws a circle of radius  $r$  with center at the point  $(x, y)$ .

*arc* draws an arc of a circle with center at the point  $(x, y)$  between the points  $(x0, y0)$  and  $(x1, y1)$ .

String arguments to *label* and *linemod* are terminated by nulls and do not contain new-lines.

See *plot(F)* for a description of the effect of the remaining functions.

The library files listed below provide several flavors of these routines.

## Files

---

|                    |                                               |
|--------------------|-----------------------------------------------|
| /usr/lib/libplot.a | produces output for <i>tplot</i> (1G) filters |
| /usr/lib/lib300.pa | for DASI 300                                  |
| /usr/lib/lib300.a  | for DASI 300s                                 |
| /usr/lib/lib450.a  | for DASI 450                                  |
| /usr/lib/lib4014.a | for TEKTRONIX 4014                            |

## See Also

---

*plot(F)*, *graph(ADM)*, *stat(F)*, *tplot(ADM)*.

## Warnings

---

In order to compile a program containing these functions in *file.c*, it is necessary to use “*cc file.c -lplot*”.

In order to execute it, it is necessary to use “*a.out | tplot*”.

The above routines use `<stdio.h>`, which causes them to increase the size of programs, not otherwise using standard I/O more than might be expected.

## poll

---

### STREAMS input/output multiplexing

### Syntax

---

```
#include <stropts.h>
#include <poll.h>

int poll(fds, nfds, timeout)
struct pollfd fds[];
unsigned long nfds;
int timeout;
```

### Description

---

The *poll* system call provides users with a mechanism for multiplexing input/output over a set of file descriptors that reference open *streams* (see *Intro(S)*) or any driver that supports the *select(S)* system call. The *poll* system call identifies those *streams* on which a user can send or receive messages, or on which certain events have occurred. A user can receive messages using *read(S)* or *getmsg(S)* and can send messages using *write(S)* and *putmsg(S)*. Certain *ioctl(S)* calls, such as *I\_RECVFD* and *I\_SENDFD*, can also be used to receive and send messages.

*fds* specifies the file descriptors to be examined and the events of interest for each file descriptor. It is a pointer to an array with one element for each open file descriptor of interest. The array's elements are *pollfd* structures which contain the following members:

|                |                        |
|----------------|------------------------|
| int fd;        | /* file descriptor */  |
| short events;  | /* requested events */ |
| short revents; | /* returned events */  |

where *fd* specifies an open file descriptor and *events* and *revents* are bitmasks constructed by or-ing any combination of the following event flags:

- |         |                                                                                                                                                                                                                                                             |
|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| POLLIN  | A non-priority or file descriptor passing message (see <i>I_RECVFD</i> ) is present on the <i>stream head</i> read queue. This flag is set even if the message is of zero length. In <i>revents</i> , this flag is mutually exclusive with <i>POLLPRI</i> . |
| POLLPRI | A priority message is present on the <i>stream head</i> read queue. This flag is set even if the message is of zero length. In <i>revents</i> , this flag is mutually exclusive with <i>POLLIN</i> .                                                        |

- POLLOUT** The first downstream write queue in the *stream* is not full. Priority control messages can be sent (see *putmsg*) at any time.
- POLLERR** An error message has arrived at the *stream head*. This flag is only valid in the *revents* bitmask; it is not used in the *events* field.
- POLLHUP** A hangup has occurred on the *stream*. This event and **POLLOUT** are mutually exclusive; a *stream* can never be writable if a hangup has occurred. However, this event and **POLLIN** or **POLLPRI** are not mutually exclusive. This flag is only valid in the *revents* bitmask; it is not used in the *events* field.
- POLLNVAL** The specified *fd* value does not belong to an open *stream*. This flag is only valid in the *revents* field; it is not used in the *events* field.

For each element of the array pointed to by *fds*, *poll* examines the given file descriptor for the event(s) specified in *events*. The number of file descriptors to be examined is specified by *nfds*. If *nfds* exceeds *NOFILES*, the system limit of open files (see *ulimit(S)*), *poll* will fail.

If the value *fd* is less than zero, *events* is ignored and *revents* is set to 0 in that entry on return from *poll*.

The results of the *poll* query are stored in the *revents* field in the *pollfd* structure. Bits are set in the *revents* bitmask to indicate which of the requested events are true. If none are true, none of the specified bits is set in *revents* when the *poll* call returns. The event flags **POLLHUP**, **POLLERR**, and **POLLNVAL** are always set in *revents* if the conditions they indicate are true; this occurs even though these flags were not present in *events*.

If none of the defined events have occurred on any selected file descriptor, *poll* waits at least *timeout* msec for an event to occur on any of the selected file descriptors. On a computer where millisecond timing accuracy is not available, *timeout* is rounded up to the nearest legal value available on that system. If the value *timeout* is 0, *poll* returns immediately. If the value of *timeout* is -1, *poll* blocks until a requested event occurs or until the call is interrupted. The *poll* system call is not affected by the *O\_NDELAY* flag.

The *poll* system call fails if one or more of the following is true:

- [EAGAIN]** Allocation of internal data structures failed but request should be attempted again.
- [EFAULT]** Some argument points outside the allocated address space.



- [EINTR] A signal was caught during the *poll* system call.
- [EINVAL] The argument *nfds* is less than zero, or *nfds* is greater than NOFiles.

## See Also

---

getmsg(S), intro(S), putmsg(S), read(S), write(S), select(S),  
*STREAMS Primer*,  
*STREAMS Programmer's Guide*

## Diagnostics

---

Upon successful completion, a non-negative value is returned. A positive value indicates the total number of file descriptors that has been selected (that is, file descriptors for which the *revents* field is non-zero). A value of 0 indicates that the call timed out and no file descriptors have been selected. Upon failure, a value of -1 is returned, and *errno* is set to indicate the error.

## Note

---

The *poll* system call also works on devices that support the *select* system call, such as the event driver and the console driver.

## Standards Conformance

---

*poll* is conformant with:  
AT&T SVID Issue 2, Select Code 307-127.

## **popen, pclose**

---

initiate pipe to/from a process

### **Syntax**

---

```
#include <stdio.h>
```

```
FILE *popen (command, type)
const char *command; char *type;
```

```
int pclose (stream)
FILE *stream;
```

### **Description**

---

The *popen* function creates a pipe between the calling program and the command to be executed. The arguments to *popen* are pointers to null-terminated strings. *command* consists of a shell command line. *type* is an I/O mode, either *r* for reading or *w* for writing. The value returned is a stream pointer such that one can write to the standard input of the command, if the I/O mode is *w*, by writing to the file *stream*; and one can read from the standard output of the command, if the I/O mode is *r*, by reading from the file *stream*.

A stream opened by *popen* should be closed by *pclose*, which waits for the associated process to terminate and returns the exit status of the command.

Because open files are shared, a type *r* command may be used as an input filter and a type *w* as an output filter.

### **Example**

---

A typical call may be:

```
char *cmd = "ls *.c";
FILE *ptr;
if ((ptr = popen(cmd, "r")) != NULL)
 while (fgets(buf, n, ptr) != NULL)
 (void) printf("%s ", buf);
```

This will print in *stdout* (see *stdio* (S)) all the file names in the current directory that have a ".c" suffix.

## See Also

---

pipe(S), wait(S), fclose(S), fopen(S), stdio(S), system(S)

## Diagnostics

---

The *popen* function returns a NULL pointer if files or processes cannot be created.

The *pclose* function returns -1 if *stream* is not associated with a “*popened*” command.

## Warning

---

If the original and “*popened*” processes concurrently read or write a common file, neither should use buffered I/O. Problems with an output filter may be forestalled by careful buffer flushing, for example, with *fflush* (see *fclose*(S)).

## Standards Conformance

---

*pclose* and *popen* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



## printf, fprintf, sprintf

---

print formatted output

### Syntax

---

```
#include <stdio.h>
```

```
int printf (format , arg ...)
const char *format;
```

```
int fprintf (stream, format , arg ...)
FILE *stream;
const char *format;
```

```
int sprintf (s, format [, arg] ...)
const char *s; char *format;
```

### Description

---

The *printf* function places output on the standard output stream *stdout*. *fprintf* places output on the named output *stream*. *sprintf* places “output,” followed by the null character (`\0`), in consecutive bytes starting at *s*; it is the user's responsibility to ensure that enough storage is available. Each function returns the number of characters transmitted (not including the `\0` in the case of *sprintf*), or a negative value if an output error was encountered.

Each of these functions converts, formats, and prints its *args* under control of the *format*. The *format* is a character string that contains three types of objects: plain characters, which are simply copied to the output stream; escape sequences that represent non-graphic characters; and conversion specifications, each of which results in fetching of zero or more *args*. The results are undefined if there are insufficient *args* for the format. If the format is exhausted while *args* remain, the excess *args* are simply ignored.

Each conversion specification is introduced by the character `%`. After the `%`, the following appear in sequence:

- Zero or more *flags*, which modify the meaning of the conversion specification.

- An optional, decimal digit string specifying a minimum *field width*. If the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left-adjustment flag `‘-’`, described below, has been given) to the field width. The padding is with blanks unless the field width digit string starts with a zero, in which case the padding is with zeros.

A *precision* that gives the minimum number of digits to appear for the **d**, **i**, **o**, **u**, **x**, or **X** conversions, the number of digits to appear after the decimal point for the **e**, **E**, and **f** conversions, the maximum number of significant digits for the **g** and **G** conversion, or the maximum number of characters to be printed from a string in **s** conversion. The precision takes the form of a period (.) followed by a decimal digit string; a null digit string is treated as zero. Padding specified by the precision overrides the padding specified by the field width.

An optional **l** (ell) specifying that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion character applies to a long integer *arg*. An **l** before any other conversion character is ignored.

A character that indicates the type of conversion to be applied.

A field width or precision or both may be indicated by an asterisk (\*) instead of a digit string. In this case, an integer *arg* supplies the field width or precision. The *arg* that is actually converted is not fetched until the conversion letter is seen, so the *args* specifying field width or precision must appear *before* the *arg* (if any) to be converted. A negative field width argument is taken as a '-' flag followed by a positive field width. If the precision argument is negative, it will be changed to zero.

The flag characters and their meanings are:

- The result of the conversion will be left-justified within the field.
- + The result of a signed conversion will always begin with a sign (+ or -).

<Space>

If the first character of a signed conversion is not a sign, a blank will be prefixed to the result. This implies that if the blank and + flags both appear, the blank flag will be ignored.

- # This flag specifies that the value is to be converted to an "alternate form." For **c**, **d**, **i**, **s**, and **u** conversions, the flag has no effect. For **o** conversion, it increases the precision to force the first digit of the result to be a zero. For **x** or **X** conversion, a non-zero result will have **0x** or **0X** prefixed to it. For **e**, **E**, **f**, **g**, and **G** conversions, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point appears in the result of these conversions only if a digit follows it). For **g** and **G** conversions, trailing zeroes will *not* be removed from the result (which they normally are).

The conversion characters and their meanings are:



- d,i,o,u,x,X** The integer *arg* is converted to signed decimal (**d** or **i**), unsigned octal, (**o**), decimal (**u**), or hexadecimal notation (**x** or **X**), respectively; the letters **abcdef** are used for **x** conversion and the letters **ABCDEF** for **X** conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeroes. The default precision is 1. The result of converting a zero value with a precision of zero is a null string.
- f** The float or double *arg* is converted to decimal notation in the style "[-.ddd.ddd," where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, six digits are output; if the precision is explicitly 0, no decimal point appears.
- e,E** The float or double *arg* is converted in the style "[-.ddde±ddd," where there is one digit before the decimal point and the number of digits after it is equal to the precision; when the precision is missing, six digits are produced; if the precision is zero, no decimal point appears. The **E** format code will produce a number with **E** instead of **e** introducing the exponent. The exponent always contains at least two digits.
- g,G** The float or double *arg* is printed in style **f** or **e** (or in style **E** in the case of a **G** format code), with the precision specifying the number of significant digits. The style used depends on the value converted: style **e** will be used only if the exponent resulting from the conversion is less than -4 or greater than the precision. Trailing zeroes are removed from the result; a decimal point appears only if it is followed by a digit.
- c** The character *arg* is printed.
- s** The *arg* is taken to be a string (character pointer) and characters from the string are printed until a null character (**\0**) is encountered or the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first null character are printed. A **NULL** value for *arg* will yield undefined results.
- %** Print a **%**; no argument is converted.



In printing floating point types (float and double), if the exponent is 0x7FF and the mantissa is not equal to zero, then the output is

`[-]NaN0xdddddddd`

where 0xdddddddd is the hexadecimal representation of the leftmost 32 bits of the mantissa. If the mantissa is zero, the output is

`[±]inf.`

In no case does a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Characters generated by *printf* and *fprintf* are printed as if *putc*(S) had been called.

## Examples

---

To print a date and time in the form "Sunday, July 3, 10:02," where *weekday* and *month* are pointers to null-terminated strings:

```
printf("%s, %s %i, %d:%.2d", weekday, month, day, hour, min);
```

To print  $\pi$  to 5 decimal places:

```
printf("pi = %.5f", 4 * atan(1.0));
```

## See Also

---

`ecvt`(S), `putc`(S), `scanf`(S), `stdio`(S)

## Standards Conformance

---

*fprintf*, *printf* and *sprintf* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

## proctl

controls active processes or process groups

### Syntax

```
#include <sys/proctl.h>
```

```
int proctl(pid, command, arg)
int pid, command;
char *arg;
```

### Description

*proctl* performs a variety of functions on active processes or process groups. It has the same form as the *ioctl*(S) system call, except that a process ID (*pid*) is substituted for a file descriptor as the first parameter.

*command* is an integer mnemonic, specifying the action to be taken, and *arg* is a pointer to a data structure which defines the parameters associated with the *command* if necessary.

If *pid* is greater than zero (0), the *command* affects the process whose process ID is equal to *pid*. *pid* may be 1.

If *pid* is zero, the command is sent to all processes, except processes 0 and 1 whose process group ID is equal to the process group ID of the sender.

If *pid* is -1 and the effective user ID of the sender is not the super-user, the command is sent to all processes, except processes 0 and 1 whose real user ID is equal to the effective user ID of the sender.

If *pid* is -1 and the effective user ID of the sender is super-user, the command is sent to all processes except processes 0 and 1.

If *pid* is negative but not -1, a signal is sent to all processes whose process group ID is equal to the absolute value of *pid*.

*proctl* will fail if one or more of the following are true:

- |          |                                                                                                                                                                |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [EINVAL] | <i>command</i> or <i>arg</i> is not valid.                                                                                                                     |
| [ESRCH]  | No process can be found to match the specified <i>pid</i> .                                                                                                    |
| [EPERM]  | The user ID of the sending process is not super-user, and its real or effective user ID does not match the real or effective user ID of the receiving process. |

[ENOMEM]      The program has requested more memory than is available.

[EFAULT]      *arg* is not a valid address.

## Memory Restrictions

---

*exec(S)* may fail when the required physical memory is larger than the available swap space. This restriction may be lifted using one of the following *proctl* commands:

### PRHUGEX

Allows programs to be executed by this process even if they exceed the available swap space. Such programs must still fit in the available physical memory and the caller's effective user ID must be super-user. Such HUGE processes are locked in memory to prevent them from being swapped. Processes that are marked HUGE with this system call but are not greater than the size of the swapper behave normally but can expand into a HUGE, locked process.

### PRNORMEX

Makes a process unable to *exec(S)* HUGE programs. This call may be executed by any user. If an attempt is made to classify a process as normal using the PRNORMEX call when the process is already too big to swap, the *proctl* call will fail, returning EINVAL.

For example, you can use the following code to allow a process to be executed even if it exceeds the available memory swapping space:

```
if (argc < 2) {
 fputs ("usage: runbig command arg ...\n", stderr);
 exit(2);
}
argv[argc] = 0;

if (proctl(getpid(), PRHUGEX, (char *) 0) < 0) {
 perror ("runbig");
 exit(1);
}
```

## Return Value

---

If an error has occurred, a value of -1 is returned and *errno* is set to indicate the error.



**See Also**

---

exec(S), ioctl(S), kill(S)

**Notes**

---

This function must be linked with the linker option **-lx**.

## profil

---

execution time profile

### Syntax

---

```
void profil (buff, bufsiz, offset, scale)
void (* offset) ();
char *buff;
unsigned bufsiz, scale;
```

### Description

---

*buff* points to an area of core whose length (in bytes) is given by *bufsiz*. After this call, the user's program counter (*pc*) is examined each clock tick. Then the value of *offset* is subtracted from it, and the remainder multiplied by *scale*. If the resulting number corresponds to an entry inside *buff*, that entry is incremented. An entry is defined as a series of bytes with length *sizeof(short)*.

The scale is interpreted as an unsigned, fixed-point fraction with binary point at the left: 0177777 (octal) gives a 1-1 mapping of *pc*'s to entries in *buff*; 077777 (octal) maps each pair of instruction entries together. 02(octal) maps all instructions onto the beginning of *buff* (producing a non-interrupting core clock).

Profiling is turned off by giving a *scale* of 0 or 1. It is rendered ineffective by giving a *bufsiz* of 0. Profiling is turned off when an *exec* is executed, but remains on in child and parent both after a *fork*. Profiling will be turned off if an update in *buff* would cause a memory fault.

### See Also

---

prof(CP), times(S), monitor(S)

### Standards Conformance

---

*profil* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## ptrace

### process trace

### Syntax

```
int ptrace (request, pid, addr, data);
int request, pid, data;
```

### Description

The *ptrace* system call provides a means by which a parent process may control the execution of a child process. Its primary use is for the implementation of breakpoint debugging (see *sdb*(CP)). The child process behaves normally until it encounters a signal (see *signal*(S) for the list), at which time it enters a stopped state and its parent is notified via *wait*(S). When the child is in the stopped state, its parent can examine and modify its "core image" using *ptrace*. Also, the parent can cause the child either to terminate or continue, with the possibility of ignoring the signal that caused it to stop. The data type of the argument *addr* depends upon the particular request given to *ptrace*.

The *request* argument determines the precise action to be taken by *ptrace* and is one of the following:

- 0 This request must be issued by the child process if it is to be traced by its parent. It turns on the child's trace flag that stipulates that the child should be left in a stopped state upon receipt of a signal rather than the state specified by *func* (see *signal*(S)). The *pid*, *addr*, and *data* arguments are ignored, and a return value is not defined for this request. Peculiar results will ensue if the parent does not expect to trace the child.

The remainder of the requests can only be used by the parent process. For each, *pid* is the process ID of the child. The child must be in a stopped state before these requests are made.

- 1, 2 With these requests, the word at location *addr* in the address space of the child is returned to the parent process. If I and D space are separated, request 1 returns a word from I space, and request 2 returns a word from D space. If I and D space are not separated, either request 1 or request 2 may be used with equal results. The *data* argument is ignored.
- 3 With this request, the word at location *addr* in the child's USER area in the system's address space (see <*sys/user.h*>) is returned to the parent process. The *data* argument is



ignored. This request will fail if *addr* is outside the USER area, in which case a value of -1 is returned to the parent process and the parent's *errno* is set to EIO.

- 4, 5 With these requests, the value given by the *data* argument is written into the address space of the child at location *addr*. If I and D space are separated, request 4 writes a word into I space, and request 5 writes a word into D space. If I and D space are not separated, either request 4 or request 5 may be used with equal results. Upon successful completion, the value written into the address space of the child is returned to the parent. These two requests will fail if *addr* is a location in a pure procedure space and another process is executing in that space. Upon failure a value of -1 is returned to the parent process and the parent's *errno* is set to EIO.
- 6 With this request, a few entries in the child's USER area can be written. *Data* gives the value that is to be written and *addr* is the location of the entry. The few entries that can be written are all registers.

On the 80386, the *ptrace* system call can be used to modify the debug registers.

The 80386 debug registers are used to specify an address to monitor in a user process. Any access to this location by the user process will deliver a **SIGTRAP** (see *signal(S)*) to the user process and possibly restart the parent process.

The 80386 debug registers can be accessed by using the 3 or 6 options of the *ptrace* system call to read or write a traced-process's u-area. The file `<sys/debugreg.h>` should be included in the parent process that wants to control the debug registers. This header file defines bit masks that describe the debug-registers in the `u_debugreg[]` array in the u-area.

The debug registers numbered `u.u_debugreg[DR_FIRSTADDR]` (`%dr0`) to `u.u_debugreg[DR_LASTADDR]` (`%dr3`) contain process addresses which will be monitored according to the instructions provided in `u.u_debugreg[DR_CONTROL]` (`%dr7`). Only the `DR_LOCAL_ENABLE_MASK` and the various read/write and length bits in `u.u_debugreg[DR_CONTROL]` can be set. Setting `DR_LOCAL_SLOWDOWN` to slow down processing is also highly recommended. The setting of all other bits is undefined and should be set to zero to ensure compatibility with future Intel processors.

In the process being debugged, these registers are automatically loaded before entering user-mode (privilege level 3) and cleared before entering the system for any reason. In

System V Release 3.2, if the location specified by a debug-register is accessed during a system call, core-dump, or interrupt service, no trap will ensue.

- 7 This request causes the child to resume execution. If the *data* argument is 0, all pending signals including the one that caused the child to stop are canceled before it resumes execution. If the *data* argument is a valid signal number, the child resumes execution as if it had incurred that signal, and any other pending signals are canceled. The *addr* argument must be equal to 1 for this request. Upon successful completion, the value of *data* is returned to the parent. This request will fail if *data* is not 0 or a valid signal number, in which case a value of -1 is returned to the parent process and the parent's *errno* is set to EIO.
- 8 This request causes the child to terminate with the same consequences as *exit*(S).
- 9 This request sets the trace bit in the Processor Status Word of the child and then executes the same steps as listed above for request 7. The trace bit causes an interrupt upon completion of one machine instruction. This effectively allows single stepping of the child.

To forestall possible fraud, *ptrace* inhibits the set-user-ID facility on subsequent *exec*(S) calls. If a traced process calls *exec*, it will stop before executing the first instruction of the new image showing signal SIGTRAP.

## General Errors

---

The *ptrace* system call will in general fail if the child process is running under *i286emul*(CP) or one or more of the following is true:

- |         |                                                                                                       |
|---------|-------------------------------------------------------------------------------------------------------|
| [EIO]   | <i>request</i> is an illegal number.                                                                  |
| [ESRCH] | <i>pid</i> identifies a child that does not exist or has not executed a <i>ptrace</i> with request 0. |

## See Also

---

*sdb*(CP), *exec*(S), *signal*(S), *wait*(S)

## Standards Conformance

---

*ptrace* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



## **putc, putchar, fputc, putw**

---

put character or word on a stream

### **Syntax**

---

```
#include <stdio.h>
```

```
int putc (c, stream)
```

```
int c;
```

```
FILE *stream;
```

```
int putchar (c)
```

```
int c;
```

```
int fputc (c, stream)
```

```
int c;
```

```
FILE *stream;
```

```
int putw (w, stream)
```

```
int w;
```

```
FILE *stream;
```

### **Description**

---

The *putc* function writes the character *c* onto the output *stream* (at the position where the file pointer, if defined, is pointing). *putchar(c)* is defined as *putc(c, stdout)*. *putc* and *putchar* are macros.

*fputc* behaves like *putc*, but is a function rather than a macro. *fputc* runs more slowly than *putc*, but it takes less space per invocation and its name can be passed as an argument to a function.

*putw* writes the word (that is, integer) *w* to the output *stream* (at the position at which the file pointer, if defined, is pointing). The size of a word is the size of an integer and varies from machine to machine. *putw* neither assumes nor causes special alignment in the file.

### **See Also**

---

*fclose(S)*, *ferror(S)*, *fopen(S)*, *fread(S)*, *printf(S)*, *puts(S)*, *setbuf(S)*, *stdio(S)*



## Diagnostics

---

On success, these functions (with the exception of *putw*) each return the value they have written. (*putw* returns *ferror (stream)*.) On failure, they return the constant EOF. This will occur if the file *stream* is not open for writing or if the output file cannot grow. Because EOF is a valid integer, *ferror(S)* should be used to detect *putw* errors.

## Notes

---

Because it is implemented as a macro, *putc* evaluates a *stream* argument more than once. In particular, *putc(c, \*f++)* doesn't work sensibly. *Fputc* should be used instead.

Because of possible differences in word length and byte ordering, files written using *putw* are machine-dependent, and may not be read using *getw* on a different processor.

## Standards Conformance

---

*fputc*, *putc* and *putchar* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

*putw* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

and The X/Open Portability Guide II of January 1987.

## putenv

---

change or add value to environment

### Syntax

---

```
int putenv (string)
char *string;
```

### Description

---

*string* points to a string of the form "*name=value*." The *putenv* function makes the value of the environment variable *name* equal to *value* by altering an existing variable or creating a new one. In either case, the string pointed to by *string* becomes part of the environment, so altering the string will change the environment. The space used by *string* is no longer used once a new string defining *name* is passed to *putenv*.

### See Also

---

exec(S), getenv(S), malloc(S), environ(M)

### Diagnostics

---

The *putenv* function returns non-zero if it was unable to obtain enough space via *malloc* for an expanded environment, otherwise zero.

### Warnings

---

The *putenv* function manipulates the environment pointed to by *environ*, and can be used in conjunction with *getenv*. However, *envp* (the third argument to *main*) is not changed. After *putenv* is called, environmental variables are not in alphabetical order.

A potential error is to call *putenv* with an automatic variable as the argument, then exit the calling function while *string* is still part of the environment.

This routine uses *malloc*(S) to enlarge the environment.

## **Standards Conformance**

---

*putenv* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



## putmsg

---

send a message on a stream

### Syntax

---

```
#include <stropts.h>
```

```
int putmsg (fd, ctlptr, dataptr, flags)
int fd;
struct strbuf *ctlptr;
struct strbuf *dataptr;
int flags;
```

### Description

---

The *putmsg* system call creates a message (see *intro(S)*) from user specified buffer(s) and sends the message to a STREAMS file. The message may contain either a data part, a control part or both. The data and control parts to be sent are distinguished by placement in separate buffers, as described below. The semantics of each part is defined by the STREAMS module that receives the message.

*fd* specifies a file descriptor referencing an open *stream*. *ctlptr* and *dataptr* each point to a *strbuf* structure which contains the following members:

|             |                      |
|-------------|----------------------|
| int maxlen; | /* not used */       |
| int len;    | /* length of data */ |
| char *buf;  | /* ptr to buffer */  |

*ctlptr* points to the structure describing the control part, if any, to be included in the message. The *buf* field in the *strbuf* structure points to the buffer where the control information resides, and the *len* field indicates the number of bytes to be sent. The *maxlen* field is not used in *putmsg* (see *getmsg(S)*). In a similar manner, *dataptr* specifies the data, if any, to be included in the message. *flags* may be set to the values 0 or RS\_HIPRI and is used as described below.

To send the data part of a message, *dataptr* must be non-NULL and the *len* field of *dataptr* must have a value of 0 or greater. To send the control part of a message, the corresponding values must be set for *ctlptr*. No data (control) part will be sent if either *dataptr* (*ctlptr*) is NULL or the *len* field of *dataptr* (*ctlptr*) is set to -1.

If a control part is specified, and *flags* is set to RS\_HIPRI, a priority message is sent. If *flags* is set to 0, a non-priority message is sent. If no control part is specified, and *flags* is set to RS\_HIPRI, *putmsg* fails and sets *errno* to EINVAL. If no control part and no data part are specified, and *flags* is set to 0, no message is sent, and 0 is returned.

For non-priority messages, *putmsg* will block if the *stream* write queue is full due to internal flow control conditions. For priority messages, *putmsg* does not block on this condition. For non-priority messages, *putmsg* does not block when the write queue is full and O\_NDELAY is set. Instead, it fails and sets *errno* to EAGAIN.

The *putmsg* system call also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks in the *stream*, regardless of priority or whether O\_NDELAY has been specified. No partial message is sent.

The *putmsg* system call fails if one or more of the following is true:

- [EAGAIN]      A non-priority message was specified, the O\_NDELAY flag is set, and the *stream* write queue is full due to internal flow control conditions.
- [EAGAIN]      Buffers could not be allocated for the message that was to be created.
- [EBADF]      *fd* is not a valid file descriptor open for writing.
- [EFAULT]      *ctlptr* or *dataptr* points outside the allocated address space.
- [EINTR]      A signal was caught during the *putmsg* system call.
- [EINVAL]      An undefined value was specified in *flags*, or *flags* is set to RS\_HIPRI and no control part was supplied.
- [EINVAL]      The *stream* referenced by *fd* is linked below a multiplexer.
- [ENOSTR]      A *stream* is not associated with *fd*.
- [ENXIO]      A hangup condition was generated downstream for the specified *stream*.
- [ERANGE]      The size of the data part of the message does not fall within the range specified by the maximum and minimum packet sizes of the topmost *stream* module. This value is also returned if the control part of the message is larger than the maximum configured size of the control part of a message, or if the data part of a message is larger than the maximum configured size of the data part of a message.

A *putmsg* also fails if a STREAMS error message had been processed by the *stream* head before the call to *putmsg*. The error returned is the value contained in the STREAMS error message.

## See Also

---

intro(S), read(S), getmsg(S), poll(S), write(S),  
*STREAMS Primer*,  
*STREAMS Programmer's Guide*

## Diagnostics

---

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## Standards Conformance

---

*putmsg* is conformant with:

AT&T SVID Issue 2, Select Code 307-127.



## putpwent

---

write password file entry

### Syntax

---

```
#include <pwd.h>
```

```
int putpwent (p, f)
struct passwd *p;
FILE *f;
```

### Description

---

The *putpwent* function is the inverse of *getpwent*(S). Given a pointer to a *passwd* structure created by *getpwent* (or *getpwuid* or *getpwnam*), *putpwent* writes a line on the stream *f*, which matches the format of */etc/passwd*.

### See Also

---

*getpwent*(S)

### Diagnostics

---

The *putpwent* function returns non-zero if an error was detected during its operation, otherwise zero.

### Warning

---

The above routine uses *<stdio.h>*, which causes it to increase the size of programs, not otherwise using standard I/O, more than might be expected.

### Standards Conformance

---

*putpwent* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## puts, fputs

---

put a string on a stream

### Syntax

---

```
#include <stdio.h>
```

```
int puts (s)
char *s;
```

```
int fputs (s, stream)
const char *s;
FILE *stream;
```

### Description

---

The *puts* function writes the null-terminated string pointed to by *s*, followed by a new-line character, to the standard output stream *stdout*.

*fputs* writes the null-terminated string pointed to by *s* to the named output *stream*.

Neither function writes the terminating null character.

### See Also

---

ferror(S), fopen(S), fread(S), printf(S), putc(S), stdio(S)

### Diagnostics

---

Both routines return EOF on error. This will happen if the routines try to write on a file that has not been opened for writing.

### Notes

---

The *puts* function appends a new-line character while *fputs* does not.

## Standards Conformance

---

*fputs* and *puts* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.



## **pw\_mapping: pw\_nametoid, pw\_idtoname, gr\_nametoid, gr\_idtoname**

---

map between user and group names and IDs

### **Syntax**

---

```
int pw_nametoid(name)
char *name;
```

```
char *pw_idtoname(id)
int id;
```

```
int gr_nametoid(name)
char *name;
```

```
char *gr_idtoname(id)
int id;
```

### **Description.**

---

These routines provide an efficient mapping between user and group names and identifiers (IDs). They maintain a separate binary database, which is automatically updated each time the routines encounter a changed */etc/passwd* or */etc/group* file. Also, these routines do not interfere with the behavior of the *getpwent*(S) and *getgrent*(S) routines, and are thus used by the Protected Database routines, which must frequently convert between names and identifiers but may not disrupt an application's use of password mapping routines.

These routines are much more efficient than the *getpwent*(S) and *getgrent*(S) routines, in that they do not parse the underlying files and keep the database in memory after the first use. Thus, savings are much greater for the second and subsequent lookups. The binary databases are stored in the */etc/auth/system* directory, which is maintained by the Auth protected subsystem.

### **Diagnostics**

---

Routines returning character strings return NULL on failure, and a pointer to a string in an internal (to the routines) memory area containing the user or group name on success. Routines returning integers return a non-negative user or group ID on success, or a -1 on failure.

## See Also

---

getpwent(S), getgrent(S), getprpwent(S)

## Files

---

/etc/auth/system/pw\_id\_map  
/etc/auth/system/gr\_id\_map  
/etc/passwd  
/etc/group

## Notes

---

Programs using these routines must be linked with *-lprot*.

## qsort

---

quicker sort

### Syntax

---

```
void qsort ((char *) base, nel, sizeof (*base), compar)
unsigned nel;
int (*compar)();
```

### Description

---

The *qsort* function is an implementation of the quicker-sort algorithm. It sorts a table of data in place.

*base* points to the element at the base of the table. *nel* is the number of elements in the table. *compar* is the name of the comparison function, which is called with two arguments that point to the elements being compared. The comparison function must return an integer less than, equal to, or greater than zero, according to whether the first argument is to be considered as less than, equal to, or greater than the second argument.

### See Also

---

bsearch(S), lsearch(S), string(S), sort(C)

### Notes

---

The pointer to the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

The order in the output of two items which compare as equal is unpredictable.



## Standards Conformance

---

*qsort* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

## raise

---

send signal *sig* to execution program

### Syntax

---

```
#include <signal.h>
int raise(int sig);
```

### Description

---

The *raise* function sends the signal *sig* to the execution program.

### Return Value

---

The *raise* function returns a zero if successful, nonzero if unsuccessful.

### See Also

---

signal(S), sigset(S)

### Standards Conformance

---

*raise* is conformant with:

ANSI X3.159-198X C Language Draft Standard, May 13, 1988.

## rand, srand

---

simple random-number generator

### Syntax

---

**int** rand ( )

**void** srand (seed)  
**unsigned** seed;

### Description

---

The *rand* function uses a multiplicative congruential random-number generator with period  $2^{32}$  that returns successive pseudo-random numbers in the range from 0 to  $2^{15}-1$ .

The *srand* function can be called at any time to reset the random-number generator to a random starting point. The generator is initially seeded with a value of 1.

### See Also

---

drand48(S)

### Notes

---

The spectral properties of *rand* are limited. The *drand48*(S) function provides a much better, though more elaborate, random-number generator.

The following functions define the semantics of the functions *rand* and *srand*.

```
static unsigned long int next = 1;
int rand()
{
 next = next * 1103515245 + 12345;
 return ((unsigned int)(next/65536) % 32768);
}
void srand(seed)
unsigned int seed;
{
 next = seed;
}
```



Specifying the semantics makes it possible to reproduce the behavior of programs that use pseudo-random sequences. This facilitates the testing of portable applications in different implementations.

## **Standards Conformance**

---

*rand* and *srand* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

## randomword

---

generate a pronounceable password

### Syntax

---

```
#include <sys/types.h>
#include <sys/security.h>
#include <sys/audit.h>
#include <prot.h>
```

```
int randomword (word, hyphenated_word, min, max, restrict,
 seed)
char *word;
char *hyphenated_word;
unsigned short int min;
unsigned short int max;
int restrict;
long seed;
```

### Description

---

*randomword* places a null-terminated random pronounceable password into the user-supplied space *word* and returns the length of the generated word. The hyphenated version of the word is placed in *hyphenated\_word*.

The *min* and *max* arguments denote the minimum and maximum lengths allowed for *word*. *Min* may equal *max*, but may not be greater than *max*, and neither may be negative. The user space pre-allocated is at least *max* for *word* and  $2*max-1$  for *hyphenated\_word*. Of course, the smaller *min* and *max* are, the less the selection of random words. For a program like *login(M)*, suggested values are 6 for *min* and 8 for *max*.

The argument *restrict* is 0 when no restrictions are placed on the generated word. It is non-zero when the words generated pass the tests of the routine *acceptable\_password* described by *accept\_pw(S)*.

The *seed* argument specifies an initial seed for the random number generator used by *randomword*. It is used by the routine only on the first time it is called; the argument is ignored on subsequent calls.

## Notes

---

The password generator relies on a random number generator that produces uniformly distributed integers. Because the password generator invokes the random number generator many times even for one word, the random number generator has to be very good. The period (distinct numbers produced given a particular seed) and number space (range of possible numbers) must both be large.

## Files

---

/etc/passwd  
/etc/group

## See Also

---

login(M), passwd(C), accept\_pw(S), drand48(S), seed(S)

## Value Added

---

*randomword* is an extension of AT&T System V provided by the Santa Cruz Operation.



## rdchk

---

checks to see if there is data to be read

### Syntax

---

```
int rdchk(fdes);
int fdes;
```

### Description

---

*rdchk* checks to see if a process will block if it attempts to read the file designated by *fdes*. *rdchk* returns 1 if the process will not block when a read is attempted or if it is the end of the file (EOF). In this context, the proper sequence of calls using *rdchk* is:

```
if(rdchk(fildes) > 0)
 read(fildes, buffer, nbytes);
```

### See Also

---

`read(S)`

### Diagnostics

---

*rdchk* returns -1 if an error occurs (e.g., EBADF), 0 if the process will block if it issues a *read* and 1 if it is okay to read. EBADF is returned if a *rdchk* is done on a semaphore file or if the file specified doesn't exist.

### Notes

---

This function must be linked with the linker option **-lx**.

## read

---

read from file

### Syntax

---

```
int read (fildes, buf, nbyte)
int fildes;
char *buf;
unsigned nbyte;
```

### Description

---

*fildes* is a file descriptor obtained from a *creat*(S), *open*(S), *dup*(S), *fcntl*(S), or *pipe*(S) system call.

The *read* system call attempts to read *nbyte* bytes from the file associated with *fildes* into the buffer pointed to by *buf*.

On devices capable of seeking, the *read* starts at a position in the file given by the file pointer associated with *fildes*. Upon return from *read*, the file pointer is incremented by the number of bytes actually read.

Devices that are incapable of seeking always read from the current position. The value of a file pointer associated with such a file is undefined.

Upon successful completion, *read* returns the number of bytes actually read and placed in the buffer; this number may be less than *nbyte* if the file is associated with a communication line (see *ioctl*(S) and *termio*(M)), or if the number of bytes left in the file is less than *nbyte* bytes. A value of 0 is returned when an end-of-file has been reached.

A *read* from a STREAMS (see *intro*(S)) file can operate in three different modes: "byte-stream" mode, "message-nondiscard" mode, and "message-discard" mode. The default is byte-stream mode. This can be changed using the *I\_SRDOPT ioctl* request, and can be tested with the *I\_GRDOPT ioctl*. In byte-stream mode, *read* will retrieve data from the *stream* until it has retrieved *nbyte* bytes, or until there is no more data to be retrieved. Byte-stream mode ignores message boundaries.

In STREAMS message-nondiscard mode, *read* retrieves data until it has read *nbyte* bytes, or until it reaches a message boundary. If the *read* does not retrieve all the data in a message, the remaining data are replaced on the *stream*, and can be retrieved by the next *read* or *getmsg*(S) call. Message-discard mode also retrieves data until it has retrieved *nbyte* bytes, or it reaches a message boundary. However,

unread data remaining in a message after the *read* returns are discarded and are not available for a subsequent *read* or *getmsg*.

When attempting to read from a regular file with mandatory file/record locking set (see *chmod(S)*), and there is a blocking (that is, owned by another process) write lock on the segment of the file to be read:

If *O\_NDELAY* is set, the read will return a -1 and set *errno* to *EAGAIN*.

If *O\_NDELAY* is clear, the read will sleep until the blocking record lock is removed.

When attempting to read from an empty pipe (or FIFO):

If *O\_NDELAY* is set, the read will return a 0.

If *O\_NDELAY* is clear, the read will block until data is written to the file or the file is no longer open for writing.

When attempting to read a file associated with a tty that has no data currently available:

If *O\_NDELAY* is set, the read will return a 0.

If *O\_NDELAY* is clear, the read will block until data becomes available.

When attempting to read a file associated with a *stream* that has no data currently available:

If *O\_NDELAY* is set, the read will return a -1 and set *errno* to *EAGAIN*.

If *O\_NDELAY* is clear, the read will block until data becomes available.

When reading from a STREAMS file, handling of zero-byte messages is determined by the current read mode setting. In byte-stream mode, *read* accepts data until it has read *nbyte* bytes, or until there is no more data to read, or until a zero-byte message block is encountered. The *read* system call then returns the number of bytes read, and places the zero-byte message back on the *stream* to be retrieved by the next *read* or *getmsg*. In the two other modes, a zero-byte message returns a value of 0 and the message is removed from the *stream*. When a zero-byte message is read as the first message on a *stream*, a value of 0 is returned regardless of the read mode.

A *read* from a STREAMS file can only process data messages. It cannot process any type of protocol message and will fail if a protocol message is encountered at the *stream head*.



The *read* system call will fail if one or more of the following are true:

- [EAGAIN] Mandatory file/record locking was set, O\_NDELAY was set, and there was a blocking record lock.
- [EAGAIN] Total amount of system memory available when reading via raw IO is temporarily insufficient.
- [EAGAIN] No message waiting to be read on a *stream* and O\_NDELAY flag set.
- [EBADF] *fildev* is not a valid file descriptor open for reading.
- [EBADMSG] Message waiting to be read on a *stream* is not a data message.
- [EDEADLK] The read was going to go to sleep and cause a deadlock situation to occur.
- [EFAULT] *buf* points outside the allocated address space.
- [EINTR] A signal was caught during the *read* system call.
- [EIO] A physical I/O error has occurred.
- [ENXIO] The device associated with the file-descriptor is a block-special or character-special file, and the value of the file-pointer is out of range.
- [EINVAL] Attempted to read from a *stream* linked to a multiplexer.
- [ENOLCK] The system record lock table was full, so the read could not go to sleep until the blocking record lock was removed.
- [ENOLINK] *fildev* is on a remote machine and the link to that machine is no longer active.

A *read* from a STREAMS file will also fail if an error message is received at the *stream head*. In this case, *errno* is set to the value returned in the error message. If a hangup occurs on the *stream* being read, *read* will continue to operate normally until the *stream head* read queue is empty. Thereafter, it will return 0.

## See Also

---

*creat*(S), *dup*(S), *fcntl*(S), *ioctl*(S), *intro*(S), *open*(S), *pipe*(S), *getmsg*(S).

## Diagnostics

---

Upon successful completion a non-negative integer is returned indicating the number of bytes actually read. Otherwise, a -1 is returned, and *errno* is set to indicate the error.

## Standards Conformance

---

*read* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

## regcmp, regex

compile and execute regular expression

### Syntax

```
char *regcmp (string1 [, string2, ...], (char *)0)
char *string1, *string2, ...;
```

```
char *regex (re, subject[, ret0, ...])
char *re, *subject, *ret0, ...;
```

```
extern char * __loc1;
```

### Description

The *regcmp* function compiles a regular expression (consisting of the concatenated arguments) and returns a pointer to the compiled form. The *malloc*(S) function is used to create space for the compiled form. It is the user's responsibility to free unneeded space so allocated. A NULL return from *regcmp* indicates an incorrect argument. *regcmp*(CP) has been written to generally preclude the need for this routine at execution time.

*regex* executes a compiled pattern against the subject string. Additional arguments are passed to receive values back. *regex* returns NULL on failure or a pointer to the next unmatched character on success. A global character pointer *\_\_loc1* points to where the match began. *regcmp* and *regex* were mostly borrowed from the editor, *ed*(C); however, the syntax and semantics have been changed slightly. The following are the valid symbols and their associated meanings.

- [ ] \* . ^ These symbols retain their meaning in *ed*(C).
- \$ Matches the end of the string; \n matches a new-line.
- Within brackets the minus means *through*. For example, [a-z] is equivalent to [abcd...xyz]. The - can appear as itself only if used as the first or last character. For example, the character class expression []- matches the characters ] and -.
- + A regular expression followed by + means *one or more times*. For example, [0-9]+ is equivalent to [0-9] [0-9]\*.
- {m} {m,u} {m,u} Integer values enclosed in {} indicate the number of times the preceding regular expression is to be applied. The value *m* is the minimum number and *u* is a number, less than 256, which



is the maximum. If only  $m$  is present (for example,  $\{m\}$ ), it indicates the exact number of times the regular expression is to be applied. The value  $\{m,\}$  is analogous to  $\{m,\text{infinity}\}$ . The plus (+) and star (\*) operations are equivalent to  $\{1,\}$  and  $\{0,\}$  respectively.

(...)\$ $n$  The value of the enclosed regular expression is to be returned. The value will be stored in the  $(n+1)$ th argument following the subject argument. At most ten enclosed regular expressions are allowed. *regex* makes its assignments unconditionally.

(...) Parentheses are used for grouping. An operator, e.g., \*, +, {}, can work on a single character or a regular expression enclosed in parentheses. For example,  $(a*(cb+)*)\$0$ .

By necessity, all the above defined symbols are special. They must, therefore, be escaped with a \ (backslash) to be used as themselves.

## Examples

---

### Example 1:

```
char *cursor, *newcursor, *ptr;
...
newcursor = regex(ptr = regcmp("^\\n", (char *)0), cursor);
free(ptr);
```

This example will match a leading new-line in the subject string pointed at by cursor.

### Example 2:

```
char ret0[9];
char *newcursor, *name;
...
name = regcmp("[A-Za-z][A-Za-z0-9]{0,7})\$0", (char *)0);
newcursor = regex(name, "012Testing345", ret0);
```

This example will match through the string "Testing3" and will return the address of the character after the last matched character (the "4"). The string "Testing3" will be copied to the character array *ret0*.

### Example 3:

```
#include "file.i"
char *string, *newcursor;
...
newcursor = regex(name, string);
```

This example applies a precompiled regular expression in **file.i** (see *regcmp*(CP)) against *string*.

## See Also

---

*regcmp*(CP), *malloc*(S), *ed*(C)

## Notes

---

The user program may run out of memory if *regcmp* is called iteratively without freeing the vectors no longer required.

## regex, regcmp

compiles and executes regular expressions.

### Syntax

```
char *regcmp(string1[,string2, ...],(char *)0);
char *string1, *string2, ...;
```

```
char *regex(re,subject[,ret0, ...]);
char *re, *subject, *ret0, ...;
extern char * __loc1;
```

### Description

The *regcmp* routine compiles a regular expression and returns a pointer to the compiled form. The *malloc*(S) routine is used to create space for the vector. It is the user's responsibility to free unneeded space so allocated. A zero return from *regcmp* indicates an incorrect argument. *regcmp*(CP) has been written to generally preclude the need for this routine at execution time.

The *regex* routine executes a compiled pattern against the subject string. Additional arguments are passed to receive values back. *regex* returns zero on failure or a pointer to the next unmatched character on success. A global character pointer *\_\_loc1* points to where the match began. *regcmp* and *regex* were derived from the editor, *ed*(C); however, the syntax and semantics have been changed slightly. The following are the valid symbols and their associated meanings.

- [ ] \* . ^ These symbols retain their current meaning.
- \$ Matches the end of the string.
- \n matches the newline.
- Within brackets the minus means *through*. For example, [a-z] is equivalent to [abcd...xyz]. The - can appear as itself only if used as the last or first character. For example, the character class expression [ ]- matches the characters ] and -.
- + A regular expression followed by + means "one or more times". For example, [0-9]+ is equivalent to [0-9][0-9]\*.
- {m} {m,u} {m,u} Integer values enclosed in {} indicate the number of times the preceding regular expression is to be applied. *m* is the minimum number and *u* is a number, less than 256, which is



the maximum. If only *m* is present (e.g., {*m*}), it indicates the exact number of times the regular expression is to be applied. {*m*,} is analogous to {*m*,infinity}. The plus (+) and star (\*) operations are equivalent to {1,} and {0,} respectively.

(...)\$*n* The value of the enclosed regular expression is to be returned. The value will be stored in the (*n*+1)th argument following the subject argument. At present, at most ten enclosed regular expressions are allowed. *regex* makes its assignments unconditionally.

(...) Parentheses are used for grouping. An operator, e.g. \*, +, {}, can work on a single character or a regular expression enclosed in parenthesis. For example, (a\*(cb+\*))\$0.

By necessity, all the above defined symbols are special. They must, therefore, be escaped to be used as themselves.

## See Also

---

ed(C), regcmp(CP), free(S), malloc(S)

## Examples

---

Example 1:

```
char *cursor, *newcursor, *ptr;
...
newcursor = regex((ptr=regcmp("\n", (char*)0)), cursor);
free(ptr);
```

This example will match a leading newline in the subject string pointed at by cursor.

Example 2:

```
char ret0[9];
char *newcursor, *name;
...
name = regcmp("[A-Za-z][A-Za-z0-9]{0,7}")$0, (char*)0);
newcursor = regex(name, "123Testing321", ret0);
```

This example will match through the string *Testing3* and will return the address of the character after the last matched character (cursor+11). The string *Testing3* will be copied to the character array *ret0*.

Example 3:

```
#include "file.i"
char *string, *newcursor;
...
newcursor = regex(name,string);
```

This example applies a precompiled regular expression in *file.i* (see *regcmp* (CP)) against *string*.

Example 4:

```
char *ptr, *newcursor;

ptr = regcmp("[a-[:i=][:digit:]]*", (char*)0);
newcursor = regex(ptr, "123CHICO321");
```

It is assumed in this example that the current locale's collation rules specify the following sequence -

**A, a, B, b, C, c, CH, Ch, ch, D, d, E, e, F, f, G, g, H, h, I, i,....**

The characters **I** and **i** are also both in the same "primary" collation group.

The following characters are all members of the **digit** ctype class -

**0, 1, 2, 3, 4, 5, 6, 7, 8, 9**

This example will match through the string "123CHIC" and return the address of the character "O" in the string.

## Notes

---

The user program may run out of memory if *regcmp* is called iteratively without freeing the vectors that are no longer required. The following user-supplied replacement for *malloc* (S) reuses the same vector saving time and space:

```
/* user's program */
...
malloc(n)
{
 static int rebuf[256];
 return &rebuf;
}
```

## regexp

---

regular expression compile and match routines

### Syntax

---

```
#define INIT <declarations>
#define GETC() <getc code>
#define PEEKC() <peekc code>
#define UNGETC(c) <ungetc code>
#define RETURN(pointer) <return code>
#define ERROR(val) <error code>

#include <regexp.h>

char *compile (instring, expbuf, endbuf, eof)
char *instring, *expbuf, *endbuf;
int eof;

int step (string, expbuf)
char *string, *expbuf;

extern char *loc1, *loc2, *locs;

extern int circf, sed, nbra;
```

### Description

---

This page describes general-purpose, regular expression matching routines in the form of *ed*(C), defined in **<regexp.h>**. Programs such as *ed*(C), *sed*(C), *grep*(C), *bs*(C), *expr*(C), etc., which perform regular expression matching use this source file. In this way, only this file need be changed to maintain regular expression compatibility.

The interface to this file is complex. Programs that include this file must have the following five macros declared before the `"#include<regexp.h>"` statement. These macros are used by the *compile* routine.

|         |                                                                                                                                                                        |
|---------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GETC()  | Return the value of the next character in the regular expression pattern. Successive calls to GETC() should return successive characters of the regular expression.    |
| PEEKC() | Return the next character in the regular expression. Successive calls to PEEKC() should return the same character [which should also be the next character returned by |



GETC()).

UNGETC(*c*)

Cause the argument *c* to be returned by the next call to GETC() [and PEEKC()]. No more than one character of pushback is ever needed, and this character is guaranteed to be the last character read by GETC(). The value of the macro UNGETC(*c*) is always ignored.

RETURN(*pointer*)

This macro is used on normal exit of the *compile* routine. The value of the argument *pointer* is a pointer to the character after the last character of the compiled regular expression. This is useful to programs which have memory allocation to manage.

ERROR(*val*)

This is the abnormal return from the *compile* routine. The argument *val* is an error number (see table below for meanings). This call should never return.

| ERROR | MEANING                               |
|-------|---------------------------------------|
| 11    | Range endpoint too large.             |
| 16    | Bad number.                           |
| 25    | “\digit” out of range.                |
| 36    | Illegal or missing delimiter.         |
| 41    | No remembered search string.          |
| 42    | \( \) imbalance.                      |
| 43    | Too many \.                           |
| 44    | More than 2 numbers given in \{ \}.   |
| 45    | } expected after \.                   |
| 46    | First number exceeds second in \{ \}. |
| 49    | [ ] imbalance.                        |
| 50    | Regular expression overflow.          |

The syntax of the *compile* routine is as follows:

`compile(instring, expbuf, endbuf, eof)`

The first parameter *instring* is never used explicitly by the *compile* routine but is useful for programs that pass down different pointers to input characters. It is sometimes used in the INIT declaration (see below). Programs which call functions to input characters or have characters in an external array can pass down a value of ((char \*) 0) for this parameter.

The next parameter *expbuf* is a character pointer. It points to the place where the compiled regular expression will be placed.

The parameter *endbuf* is one more than the highest address where the compiled regular expression may be placed. If the compiled expression cannot fit in (*endbuf-expbuf*) bytes, a call to ERROR(50) is made.

The parameter *eof* is the character which marks the end of the regular expression. For example, in *ed(C)*, this character is usually a */*.

Each program that includes this file must have a **#define** statement for *INIT*. This definition will be placed right after the declaration for the function *compile* and the opening curly brace (*{*). It is used for dependent declarations and initializations. Most often it is used to set a register variable to point to the beginning of the regular expression so that this register variable can be used in the declarations for *GETC()*, *PEEK()* and *UNGETC()*. Otherwise it can be used to declare external variables that might be used by *GETC()*, *PEEK()* and *UNGETC()*. See the example below of the declarations taken from *grep(C)*.

There are other functions in this file which perform actual regular expression matching, one of which is the function *step*. The call to *step* is as follows:

```
step(string, expbuf)
```

The first parameter to *step* is a pointer to a string of characters to be checked for a match. This string should be null-terminated.

The second parameter *expbuf* is the compiled regular expression which was obtained by a call of the function *compile*.

The function *step* returns non-zero if the given string matches the regular expression, and zero if the expressions do not match. If there is a match, two external character pointers are set as a side effect to the call to *step*. The variable set in *step* is *loc1*. This is a pointer to the first character that matched the regular expression. The variable *loc2*, which is set by the function *advance*, points to the character after the last character that matches the regular expression. Thus if the regular expression matches the entire line, *loc1* will point to the first character of *string* and *loc2* will point to the null at the end of *string*.

*Step* uses the external variable *circf* which is set by *compile* if the regular expression begins with *^*. If this is set, then *step* will try to match the regular expression to the beginning of the string only. If more than one regular expression is to be compiled before the first is executed, the value of *circf* should be saved for each compiled expression, and *circf* should be set to that saved value before each call to *step*.

The function *advance* is called from *step* with the same arguments as *step*. The purpose of *step* is to step through the *string* argument and call *advance* until *advance* returns non-zero indicating a match or until the end of *string* is reached. If one wants to constrain *string* to the beginning of the line in all cases, *step* need not be called; simply call *advance*.



When *advance* encounters a `*` or `\{ \}` sequence in the regular expression, it will advance its pointer to the string to be matched as far as possible and will recursively call itself trying to match the rest of the string to the rest of the regular expression. As long as there is no match, *advance* will back up along the string until it finds a match or reaches the point in the string that initially matched the `*` or `\{ \}`. It is sometimes desirable to stop this backing up before the initial point in the string is reached. If the external character pointer *locs* is equal to the point in the string at sometime during the backing up process, *advance* will break out of the loop that backs up and will return zero. This is used by *ed*(C) and *sed*(C) for substitutions done globally (not just the first occurrence, but the whole line) so, for example, expressions like *s/y\*/g* do not loop forever.

The additional external variables *sed* and *nbra* are used for special purposes.

## EXAMPLES

---

The following is an example of how the regular expression macros and calls look from *grep*(C):

```
#define INIT register char *sp = instring;
#define GETC() (*sp++)
#define PEEKC() (*sp)
#define UNGETC(c) (--sp)
#define RETURN(c) return;
#define ERROR(c) regerr()

#include <regexp.h>
...
 (void) compile(*argv, expbuf, &expbuf[ESIZE], '\0');
...
 if (step(linebuf, expbuf))
 succeed();
```

## See Also

---

*ed*(C), *expr*(C), *grep*(C), *regcmp*(S), *sed*(C) in the *User's/System Administrator's Reference Manual*.

## Standards Conformance

---

*regexp* is conformant with:

AT&T SVID Issue 2, Select Code 307-127.



## remove

---

removes filename

### Syntax

---

```
#include <stdio.h>
int remove (filename)
const char *filename;
```

### Description

---

The *remove* function causes the file whose name is the string pointed to by *filename* to be no longer accessible by that name. A subsequent attempt to open that file using that name will fail, unless it is created anew. If the file is open, the behavior of the *remove* function is implementation-defined.

### Return Value

---

The *remove* function returns zero if the operation succeeds, nonzero if it fails.

### Standards Conformance

---

*remove* is conformant with:

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

## rename

---

changes filename

### Syntax

---

```
#include <stdio.h>
```

```
int rename (old, new)
```

```
const char *old;
```

```
const char *new;
```

### Description

---

The *rename* function causes the file or directory whose pathname is the string pointed to by *old* to be henceforth known by the pathname given by the string pointed to by *new*. The file or directory named *old* is effectively removed. If a file or directory named by the string pointed to by *new* exists prior to the call to the *rename* function, *new* is removed.

If the *old* argument and the *new* argument both refer to links to the same existing file, the *rename*( ) function returns successfully and performs no other action.

The *old* and the *new* arguments must both point to the pathname of a file or they must both point to the pathname of a directory. If the link named by the *new* argument exists, it is removed and *old* renamed to *new*. In this case, a link named *new* exists throughout the the renaming operation and refers either to the file referred to by *new* or *old* before the operation began. Write access permission is required for both the directory containing *old* and the directory containing *new*.

If the *old* argument points to the pathname of a directory, the *new* argument cannot point to the pathname of a file. If the directory named by the *new* argument exists, it is removed and *old* renamed to *new*. In this case, a link named *new* exists throughout the renaming operation and refers either to the file referred to by *new* or *old* before the operation began. Thus, if *new* names an existing directory, it is required to be an empty directory.

The *new* pathname cannot contain a path prefix that names *old*. Write access permission is required for the directory containing *old* and the directory containing *new*. If the *old* argument points to the pathname of a directory, write access permission may be required for the directory named by *old*, and, if it exists, the directory named by *new*.

If the link named by the *new* argument exists and the file's link count becomes zero when it is removed and no process has the file open, the space occupied by the file is freed and the file is no longer accessible. If one of more processes have the file open when the last link is removed, the link is removed before *rename()* returns, but the removal of the file contents is postponed until all references to the file have been closed.

Upon successful completion, the *rename()* function marks for update the *st\_ctime* and *st\_mtime* fields of the parent directory of each file.

## Return Value

---

The *rename* function returns zero if the operation succeeds. A return value of -1 indicates that an error has occurred and an error code has been stored in *errno*.

## Diagnostics

---

If any of the following conditions occur, the *rename()* function returns -1 and sets *errno* to the corresponding value:

- |             |                                                                                                                                                                                                                                                                           |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [EACCES]    | A component of either path prefix denies search permission; or one of the directories containing <i>old</i> or <i>new</i> denies write permissions; or, write permission is required and is denied for a directory pointed to the the <i>old</i> or <i>new</i> arguments. |
| [EBUSY]     | The directory named by <i>old</i> or <i>new</i> cannot be renamed because it is being used by the system or another process and the implementation considers this to be an error.                                                                                         |
| [EEXIST]    | The link named by <i>new</i> is a directory containing entries other than dot and dot-dot.                                                                                                                                                                                |
| [ENOTEMPTY] | Same as [EEXIST].                                                                                                                                                                                                                                                         |
| [EINVAL]    | The <i>new</i> directory pathname contains a path prefix that names the <i>old</i> directory.                                                                                                                                                                             |
| [EISDIR]    | The <i>new</i> argument points to a directory and the <i>old</i> argument points to a file that is not a directory.                                                                                                                                                       |



**[ENAMETOOLONG]**

The length of the *old* or *new* argument exceeds {PATH\_MAX} or a pathname component is longer than {NAME\_MAX} while {\_POSIX\_NO\_TRUNC} is in effect.

**[ENOENT]**

The link named by the *old* argument does not exist or either *old* or *new* points to an empty string.

**[ENOSPC]**

The directory that would contain *new* cannot be extended.

**[ENOTDIR]**

A component of either path prefix is not a directory; or the *old* argument names a directory and the *new* argument names a nondirectory file.

**[EROFS]**

The requested operation requires writing in a directory on a read-only file system.

**[EXDEV]**

The links named by *new* and *old* are on different file systems and the implementation does not support links between file systems.

---

**See Also**

link(S), rmdir(S), unlink(S)

---

**Standards Conformance**

*rename* is conformant with:

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

## rmdir

---

remove a directory

### Syntax

---

```
int rmdir (path)
char *path;
```

### Description

---

*rmdir* removes the directory named by the path name pointed to by *path*. The directory must not have any entries other than "." and "..".

The named directory is removed unless one or more of the following is true:

- |           |                                                                                           |
|-----------|-------------------------------------------------------------------------------------------|
| [EINVAL]  | The current directory may not be removed.                                                 |
| [EINVAL]  | The "." entry of a directory may not be removed.                                          |
| [EEXIST]  | The directory contains entries other than those for "." and "..".                         |
| [ENOTDIR] | A component of the path prefix is not a directory.                                        |
| [ENOENT]  | The named directory does not exist.                                                       |
| [EACCES]  | Search permission is denied for a component of the path prefix.                           |
| [EACCES]  | Write permission is denied on the directory containing the directory to be removed.       |
| [EBUSY]   | The directory to be removed is the mount point for a mounted file system.                 |
| [EROFS]   | The directory entry to be removed is part of a read-only file system.                     |
| [EFAULT]  | <i>path</i> points outside the process's allocated address space.                         |
| [EIO]     | An I/O error occurred while accessing the file system.                                    |
| [ENOLINK] | <i>path</i> points to a remote machine, and the link to that machine is no longer active. |

[EMULTIHOP] Components of *path* require hopping to multiple remote machines.

In addition, a directory will not be removed when all of the following conditions are true:

- the parent directory has the sticky bit set
- the parent directory is not owned by the user
- the directory is not owned by the user
- the directory is not writable by the user
- the user is not super-user

## See Also

---

mkdir(S)

rmdir(C), rm(C), and mkdir(C).

## Diagnostics

---

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## Standards Conformance

---

*rmdir* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.



# scanf, fscanf, sscanf

---

convert formatted input

## Syntax

---

```
#include <stdio.h>
```

```
int scanf (format [, pointer] ...)
const char *format;
```

```
int fscanf (stream, format [, pointer] ...)
FILE *stream;
const char *format;
```

```
int sscanf (s, format [, pointer] ...)
char *s; const char *format;
```

## Description

---

The *scanf* function reads from the standard input stream *stdin*. *fscanf* reads from the named input *stream*. *sscanf* reads from the character string *s*. Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control string *format* described below, and a set of *pointer* arguments indicating where the converted input should be stored. The results are undefined in that there are insufficient *args* for the format. If the format is exhausted while *args* remain, the excess *args* are simply ignored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. White-space characters (blanks, tabs, new-lines, or form-feeds) which, except in two cases described below, cause input to be read up to the next non-white-space character.
2. An ordinary character (not %), which must match the next character of the input stream.
3. Conversion specifications, consisting of the character %, an optional assignment suppressing character \*, an optional numerical maximum field width, an optional l (ell) or h indicating the size of the receiving variable, and a conversion code.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by \*. The suppression of assignment provides a way of describing an input field which is to be skipped. An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted. For all descriptors except "[ " and "c", white space leading an input field is ignored.

The conversion code indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. For a suppressed field, no pointer argument is given. The following conversion codes are legal:

- % a single % is expected in the input at this point; no assignment is done.
- d a decimal integer is expected; the corresponding argument should be an integer pointer.
- u an unsigned decimal integer is expected; the corresponding argument should be an unsigned integer pointer.
- o an octal integer is expected; the corresponding argument should be an integer pointer.
- x a hexadecimal integer is expected; the corresponding argument should be an integer pointer.
- i an integer is expected; the corresponding argument should be an integer pointer. It will store the value of the next input item interpreted according to C conventions: a leading "0" implies octal; a leading "0x" implies hexadecimal; otherwise, decimal.
- n stores in an integer argument the total number of characters (including white space) that have been scanned so far since the function call. No input is consumed.
- e,f,g a floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a *float*. The input format for floating point numbers is an optionally signed string of digits, possibly containing a decimal point, followed by an optional exponent field consisting of an E or an e, followed by an optional + or - , followed by an integer.
- s a character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating \0, which will be added automatically. The input field is terminated by a white-space character.



- c a character is expected; the corresponding argument should be a character pointer. The normal skip over white space is suppressed in this case; to read the next non-space character, use `%1s`. If a field width is given, the corresponding argument should refer to a character array; the indicated number of characters is read.
- [ indicates string data and the normal skip over leading white space is suppressed. The left bracket is followed by a set of characters, which we will call the *scanset*, and a right bracket; the input field is the maximal sequence of input characters consisting entirely of characters in the scanset. The carat (^), when it appears as the first character in the scanset, serves as a complement operator and redefines the scanset as the set of all characters *not* contained in the remainder of the scanset string. There are some conventions used in the construction of the scanset. A range of characters may be represented by the construct *first-last*, thus `[0123456789]` may be expressed `[0-9]`. Using this convention, *first* must be lexically less than or equal to *last*, or else the dash will stand for itself. The dash will also stand for itself whenever it is the first or the last character in the scanset. To include the right square bracket as an element of the scanset, it must appear as the first character (possibly preceded by a carat) of the scanset, and in this case it will not be syntactically interpreted as the closing bracket. The corresponding argument must point to a character array large enough to hold the data field and the terminating `\0`, which will be added automatically. At least one character must match for this conversion to be considered successful.

The conversion characters **d**, **u**, **o**, **x** and **i** may be preceded by **l** or **h** to indicate that a pointer to **long** or to **short** rather than to **int** is in the argument list. Similarly, the conversion characters **e**, **f**, and **g** may be preceded by **l** to indicate that a pointer to **double** rather than to **float** is in the argument list. The **l** or **h** modifier is ignored for other conversion characters.

The *scanf* function conversion terminates at **EOF**, at the end of the control string, or when an input character conflicts with the control string. In the latter case, the offending character is left unread in the input stream.

The *scanf* function returns the number of successfully matched and assigned input items; this number can be zero in the event of an early conflict between an input character and the control string. If the input ends before the first conflict or conversion, **EOF** is returned.



## Examples

---

The call:

```
int n ; float x; char name[50];
n = scanf ("%d%f%s", &i, &x, name);
```

with the input line:

25 54.32E-1 thompson

will assign to *n* the value **3**, to *i* the value **25**, to *x* the value **5.432**, and *name* will contain **thompson\0** . Or:

```
int i, j; float x; char name[50];
(void) scanf ("%i%2d%f*d %[0-9] ", &j, &i, &x, name);
```

with input:

011 56789 0123 56a72

will assign **9** to *j*, **56** to *i*, **789.0** to *x*, skip **0123**, and place the string **56\0** in *name* . The next call to *getchar* (see *getc* (S)) will return **a** . Or:

```
int i, j, s, e; char name[50];
(void) scanf ("%i %i %n%s%n", &i, &j, &s, name, &e);
```

with input:

0x11 0xy johnson

will assign **17** to *i*, **0** to *j*, **6** to *s*, will place the string **xy\0** in *name*, and will assign **8** to *e* . Thus, the length of *name* is  $e - s = 2$  . The next call to *getchar* (see *getc* (S)) will return a blank.

## See Also

---

*getc*(S), *printf*(S), *stdio*(S), *strtod*(S), *strtol*(S)

## Diagnostics

---

These functions return **EOF** on end of input and a short count for missing or illegal data items.

## Notes

---

Trailing white space (including a new-line) is left unread unless matched in the control string.

## Standards Conformance

---

*fscanf*, *scanf* and *sscanf* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

## sdenter, sdleave

---

synchronizes access to a shared data segment

### Syntax

---

```
#include <sys/sd.h>
```

```
int sdenter(addr, flags)
```

```
char *addr;
```

```
int flags;
```

```
int sdleave(addr)
```

```
char *addr;
```

### Description

---

*sdenter* is used to indicate that the current process is about to access the contents of a shared data segment. *addr* is the valid return code from a previous *sdget* (S) call. The actions performed depend on the value of *flags*. *flags* values are formed by OR-ing together entries from the following list:

**SD\_NOWAIT** If another process has called *sdenter* but not *sdleave* for the indicated segment, and the segment was not created with the SD\_UNLOCK flag set, return an ENAVAIL error instead of waiting for the segment to become free.

**SD\_WRITE** Indicates that the process wants to write data to the shared data segment. A process that has attached to a shared data segment with the SD\_RDONLY flag set will not be allowed to enter with the SD\_WRITE flag set.

*sdleave* is used to indicate that the current process is done modifying the contents of a shared data segment.

Only changes made between invocations of *sdenter* and *sdleave* are guaranteed to be reflected in other processes. *sdenter* and *sdleave* are very fast; consequently, it is recommended that they be called frequently rather than leave *sdenter* in effect for any period of time. In particular, system calls should be avoided between *sdenter* and *sdleave* calls.

The *fork* system call is forbidden between calls to *sdenter* and *sdleave* if the segment was created without the SD\_UNLOCK flag.



## Return Value

---

Successful calls return 0. Unsuccessful calls return -1, and *errno* is set to indicate the error. *errno* is set to EINVAL if a process does an *sdenter* with the SD\_WRITE flag set and the segment is already attached with the SD\_RDONLY flag set. *errno* is set to ENAVAIL if the SD\_NOWAIT flag is set for *sdenter* call and the shared data segment is not free.

## See Also

---

sdget(S), sdgetv(S)

## Notes

---

This feature is a XENIX specific enhancement and may not be present on all UNIX implementations. This routine must be linked with the linker option **-lx**.

## sdget, sdfree

attaches and detaches a shared data segment.

### Syntax

```
#include <sys/sd.h>
```

```
char *sdget(path, flags, [size, mode])
```

```
char *path;
```

```
int flags, mode;
```

```
long size;
```

```
int sdfree(addr);
```

```
char *addr;
```

### Description

*sdget* attaches a shared data segment to the data space of the current process. The actions performed are controlled by the value of *flags*. *flags* values are constructed by OR-ing flags from the following list:

**SD\_RDONLY** Attach the segment for reading only.

**SD\_WRITE** Attach the segment for both reading and writing.

**SD\_CREAT** If the segment named by *path* exists and is not in use (active), this flag will have the same effect as creating a segment from scratch. Otherwise, the segment is created according to the values of *size* and *mode*. Read and write access to the segment is granted to other processes based on the permissions passed in *mode*, and functions the same as those for regular files. Execute permission is meaningless. The segment is initialized to contain all zeroes.

**SD\_UNLOCK** If the segment is created because of this call, the segment will be made so that more than one process can be between *sdenter* and *sdleave* calls.

*sdfree* detaches the current process from the shared data segment that is attached at the specified address. If the current process has done *sdenter* but not an *sdleave* for the specified segment, *sdleave* will be done before detaching the segment.

When no process remains attached to the segment, the contents of that segment disappear, and no process can attach to the segment without creating it by using the **SD\_CREAT** flag in *sdget*. *errno* is set to **EEXIST** if a process tries to create a shared data segment that exists and is

in use. *errno* is set to ENOTNAM if a process attempts an *sdget* on a file that exists but is not a shared data type.

## Notes

---

Use of the SD\_UNLOCK flag on systems without hardware support for shared data may cause severe performance degradation.

For 286 programs, it is strongly recommended that *sdget* and other shared data functions be reserved for large model programs only. Small or middle model programs that attempt to use shared data may run out of available memory. Also, due to the 286 hardware, it is not possible to enforce the read-only aspect of small model shared data. However, read-only segments are honored in large model programs.

The 386 provides a 32 bit address space, even in small model. As a result, shared data may be conveniently used without regard to the restrictions that apply to 286 programs.

*sdget* automatically increments the process's original break value to the memory location immediately after the shared data segment. This affects subsequent *sbrk* or *brk* calls which attempt to restore the original break value. In particular, attempts to restore the break value to its value before the *sdget* call causes an error.

This feature is a XENIX specific enhancement and may not be present in all UNIX implementations. This routine must be linked using the linker option *-lx*.

## Return Value

---

On successful completion, the address at which the segment was attached is returned. Otherwise, -1 is returned, and *errno* is set to indicate the error. *errno* is set to EINVAL if a process does an *sdget* on a shared data segment to which it is already attached. *errno* is set to EEXIST if a process tries to create a shared data segment that exists and is in use. *errno* is set to ENOTNAM if a process attempts an *sdget* on a file that exists but is not a shared data type.

The mode parameter must be included on the first call of the *sdget()* function.

## See Also

---

*sdenter*(S), *sdgetv*(S), *sbrk*(S)



## Notes

---

The *size* variable in *sdget* has changed from unsigned to long between UNIX Version 3.0 and UNIX System V. Although this requires that source code be modified to use a long *size* parameter when compiling with the System V libraries, an unsigned *size* parameter will still be correctly interpreted by the kernel when passed by binaries compiled with the Version 3.0 libraries.

## sdgetv, sdwaitv

---

synchronizes shared data access

### Syntax

---

```
#include <sys/sd.h>
```

```
int sdgetv(addr)
char *addr;
```

```
int sdwaitv(addr, vnum)
char *addr;
int vnum;
```

### Description

---

*sdgetv* and *sdwaitv* may be used to synchronize cooperating processes that are using shared data segments. The return value of both routines is the version number of the shared data segment attached to the process at address *addr*. The version number of a segment changes whenever some process does an *sdleave* for that segment.

*sdgetv* simply returns the version number of the indicated segment.

*sdwaitv* forces the current process to sleep until the version number for the indicated segment is no longer equal to *vnum*.

### Return Value

---

Upon successful completion, both *sdgetv* and *sdwaitv* return a positive integer that is the current version number for the indicated shared data segment. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

### See Also

---

sdenter(S), sdget(S)

### Notes

---

This routine must be linked using the linker option **-lx**.

## **seed: getseed, setseed**

---

obtain or set seed for random number generator

### **Syntax**

---

```
#include <sys/types.h>
#include <sys/macros.h>
#include <sys/security.h>
#include <sys/audit.h>
#include <sys/param.h>
#include <sys/filsys.h>
#include <prot.h>
```

```
long get_seed (prpwd)
struct pr_passwd *prpwd;
```

```
void set_seed (seed)
long seed;
```

### **Description**

---

These routines handle seed management for users of the random number generator used in the Authentication subsystem. They exist to centralize control so that multiple seed algorithms can be handled together and so that different routines do not all set an initial seed (and so here it promotes the randomness desired).

*Get\_seed* uses an automatic seed generation algorithm to choose a value to return. It currently does not use the argument, but to be compatible with more secure versions of the system, it should pass in the Protected Password of the user of interest. If no user is relevant to setting the seed, *prpwd* should be NULL.

*Set\_seed* primes the program's random number generator (using *drand48(S)*) to the *seed* passed in. The routine ignores all requests except the first one, so that the generator is not reset after its first invocation.

### **Notes**

---

The *drand48(S)* routines used on an implementation should be tested to produce good uniformity, a long period, and a large name space. Not all ports test for this, and poor performance translates into predictable results in such vital operations as password generation.



SEED (S)

SEED (S)

## See Also

---

drand48(S)

## select

---

synchronous I/O multiplexing

### Syntax

---

```
#include <sys/types.h>
#include <sys/times.h>
#include <sys/select.h>
```

```
nfound = select(nfds, readfds, writefds, exceptfds, timeout)
int nfound, nfds;
fd_set *readfds, *writefds, *exceptfds;
struct timeval *timeout;
```

```
FD_SET(fd, &fdset)
FD_CLR(fd, &fdset)
FD_ISSET(fd, &fdset)
FD_ZERO(&fdset)
int fd;
fd_set fdset;
```

### Description

---

*select* examines the I/O descriptor sets whose addresses are passed in *readfds*, *writefds*, and *exceptfds* to see if some of their descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively. The first *nfds* descriptors are checked in each set; i.e. the descriptors from 0 through *nfds*-1 in the descriptor sets are examined. On return, *select* replaces the given descriptor sets with subsets consisting of those descriptors that are ready for the requested operation. The total number of ready descriptors in all the sets is returned in *nfound*.

The descriptor sets are stored as bit fields in arrays of integers. The following macros are provided for manipulating such descriptor sets: *FD\_ZERO(&fdset)* initializes a descriptor set *fdset* to the null set. *FD\_SET(fd, &fdset)* includes a particular descriptor *fd* in *fdset*. *FD\_CLR(fd, &fdset)* removes *fd* from *fdset*. *FD\_ISSET(fd, &fdset)* is nonzero if *fd* is a member of *fdset*, zero otherwise. The behavior of these macros is undefined if a descriptor value is less than zero or greater than or equal to *FD\_SETSIZE*, which is normally at least equal to the maximum number of descriptors supported by the system.

If *timeout* is a non-zero pointer, it specifies a maximum interval to wait for the selection to complete. If *timeout* is a zero pointer, the select blocks indefinitely. To effect a poll, the *timeout* argument should be non-zero, pointing to a zero-valued *timeval* structure.

Any of *readfds*, *writefds*, and *exceptfds* may be given as zero pointers if no descriptors are of interest.

## Return Value

---

*select* returns the number of ready descriptors that are contained in the descriptor sets, or -1 if an error occurred. If the time limit expires then *select* returns 0. If *select* returns with an error, including one due to an interrupted call, the descriptor sets will be unmodified.

## Errors

---

An error return from *select* indicates:

|          |                                                                                                                                                                    |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [EBADF]  | One of the descriptor sets specified an invalid descriptor.                                                                                                        |
| [EINTR]  | A signal was delivered before the time limit expired and before any of the selected events occurred.                                                               |
| [EINVAL] | The specified time limit is invalid. One of its components is negative or too large. Or, the device driver being polled has not implemented <i>select</i> support. |

## See Also

---

read(S), write(S)

## Notes

---

*select* should probably return the time remaining from the original timeout, if any, by modifying the time value in place. This may be implemented in future versions of the system. Thus, it is unwise to assume that the timeout value will be unmodified by the *select* call.

## Credit

---

This utility was developed at the University of California at Berkeley and is used with permission.



## semctl

---

### semaphore control operations

#### Syntax

---

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semctl (semid, semnum, cmd, arg)
int semid, cmd;
int semnum;
union semun {
 int val;
 struct semid_ds *buf;
 ushort *array;
} arg;
```

#### Description

---

The *semctl* system call provides a variety of semaphore control operations as specified by *cmd*.

The following *cmds* are executed with respect to the semaphore specified by *semid* and *semnum*:

|         |                                                                                                                                                                                                  |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GETVAL  | Return the value of semval (see <i>intro</i> (S)). {READ}                                                                                                                                        |
| SETVAL  | Set the value of semval to <i>arg.val</i> . {ALTER} When this <i>cmd</i> is successfully executed, the <i>semadj</i> value corresponding to the specified semaphore in all processes is cleared. |
| GETPID  | Return the value of <i>sempid</i> . {READ}                                                                                                                                                       |
| GETNCNT | Return the value of <i>semncnt</i> . {READ}                                                                                                                                                      |
| GETZCNT | Return the value of <i>semzcnt</i> . {READ}                                                                                                                                                      |

The following *cmds* return and set, respectively, every *semval* in the set of semaphores.

- GETALL** Place *semvals* into array pointed to by *arg.array*. {READ}
- SETALL** Set *semvals* according to the array pointed to by *arg.array*. {ALTER} When this *cmd* is successfully executed the *semadj* values corresponding to each specified semaphore in all processes are cleared.

The following *cmds* are also available:

- IPC\_STAT** Place the current value of each member of the data structure associated with *semid* into the structure pointed to by *arg.buf*. The contents of this structure are defined in *intro*(S). {READ}
- IPC\_SET** Set the value of the following members of the data structure associated with *semid* to the corresponding value found in the structure pointed to by *arg.buf*:
- sem\_perm.uid**  
**sem\_perm.gid**  
**sem\_perm.mode** /\* only low 9 bits \*/

This *cmd* can only be executed by a process that has an effective user ID equal to either that of super-user, or to the value of **sem\_perm.cuid** or **sem\_perm.uid** in the data structure associated with *semid*.

- IPC\_RMID** Remove the semaphore identifier specified by *semid* from the system and destroy the set of semaphores and data structure associated with it. This *cmd* can only be executed by a process that has an effective user ID equal to either that of super-user, or to the value of **sem\_perm.cuid** or **sem\_perm.uid** in the data structure associated with *semid*.

The *semctl* system call fails if one or more of the following is true:

- [EINVAL] *semid* is not a valid semaphore identifier.
- [EINVAL] *semnum* is less than zero or greater than **sem\_nsems**.
- [EINVAL] *cmd* is not a valid command.
- [EACCES] Operation permission is denied to the calling process (see *intro*(S)).

- [ERANGE] *cmd* is **SETVAL** or **SETALL** and the value to which *semval* is to be set is greater than the system imposed maximum.
- [EPERM] *cmd* is equal to **IPC\_RMID** or **IPC\_SET** and the effective user ID of the calling process is not equal to that of super-user or to the value of **sem\_perm.cuid** or **sem\_perm.uid** in the data structure associated with *semid*.
- [EFAULT] *arg.buf* points to an illegal address.

## See Also

---

intro(S), semget(S), semop(S)

## Diagnostics

---

Upon successful completion, the value returned depends on *cmd* as follows:

|                |                               |
|----------------|-------------------------------|
| <b>GETVAL</b>  | The value of <i>semval</i> .  |
| <b>GETPID</b>  | The value of <i>sempid</i> .  |
| <b>GETNCNT</b> | The value of <i>semncnt</i> . |
| <b>GETZCNT</b> | The value of <i>semzcnt</i> . |
| All others     | A value of 0.                 |

Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## Standards Conformance

---

*semctl* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



# semget

---

get set of semaphores

## Syntax

---

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
```

```
int semget (key, nsems, semflg)
key_t key;
int nsems, semflg;
```

## Description

---

The *semget* system call returns the semaphore identifier associated with *key*.

A semaphore identifier and associated data structure and set containing *nsems* semaphores (see *intro(S)*) are created for *key* if one of the following is true:

*key* is equal to **IPC\_PRIVATE**.

*key* does not already have a semaphore identifier associated with it, and (*semflg* & **IPC\_CREAT**) is "true".

Upon creation, the data structure associated with the new semaphore identifier is initialized as follows:

**sem\_perm.cuid**, **sem\_perm.uid**, **sem\_perm.cgid**, and **sem\_perm.gid** are set equal to the effective user ID and effective group ID, respectively, of the calling process.

The low-order 9 bits of **sem\_perm.mode** are set equal to the low-order 9 bits of *semflg*.

**sem\_nsems** is set equal to the value of *nsems*.

**sem\_otime** is set equal to 0 and **sem\_ctime** is set equal to the current time.

The data structure associated with each semaphore in the set is not initialized. The function *semctl* with the command *setval* or *setall* can be used to initialize each semaphore.

The *semget* system call fails if one or more of the following is true:

- |          |                                                                                                                                                                              |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [EINVAL] | <i>nsems</i> is either less than or equal to zero or greater than the system-imposed limit.                                                                                  |
| [EACCES] | A semaphore identifier exists for <i>key</i> , but operation permission (see <i>intro(S)</i> ) as specified by the low-order 9 bits of <i>semflg</i> would not be granted.   |
| [EINVAL] | A semaphore identifier exists for <i>key</i> , but the number of semaphores in the set associated with it is less than <i>nsems</i> , and <i>nsems</i> is not equal to zero. |
| [ENOENT] | A semaphore identifier does not exist for <i>key</i> , and ( <i>semflg</i> & <b>IPC_CREAT</b> ) is "false".                                                                  |
| [ENOSPC] | A semaphore identifier is to be created, but the system-imposed limit on the maximum number of allowed semaphore identifiers system wide would be exceeded.                  |
| [ENOSPC] | A semaphore identifier is to be created, but the system-imposed limit on the maximum number of allowed semaphores system wide would be exceeded.                             |
| [EEXIST] | A semaphore identifier exists for <i>key</i> , but ( <i>semflg</i> & <b>IPC_CREAT</b> ) and ( <i>semflg</i> & <b>IPC_EXCL</b> ) are "true".                                  |

## See Also

---

*intro(S)*, *semctl(S)*, *semop(S)*

## Diagnostics

---

Upon successful completion, a non-negative integer, namely a semaphore identifier, is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## Standards Conformance

---

*semget* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## semop

---

### semaphore operations

### Syntax

---

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semop (semid, sops, nsops)
int semid;
struct sembuf **sops;
unsigned nsops;
```

### Description

---

The *semop* system call is used to automatically perform an array of semaphore operations on the set of semaphores associated with the semaphore identifier specified by *semid*. *sops* is a pointer to the array of semaphore-operation structures. *nsops* is the number of such structures in the array. The contents of each structure includes the following members:

|       |          |                           |
|-------|----------|---------------------------|
| short | sem_num; | /* semaphore number */    |
| short | sem_op;  | /* semaphore operation */ |
| short | sem_flg; | /* operation flags */     |

Each semaphore operation specified by *semop* is performed on the corresponding semaphore specified by *semid* and *sem\_num*.

*semop* specifies one of three semaphore operations as follows:

If *semop* is a negative integer, one of the following will occur: {ALTER}

If *semval* (see *intro(S)*) is greater than or equal to the absolute value of *semop*, the absolute value of *semop* is subtracted from *semval*. Also, if (*sem\_flg* & SEM\_UNDO) is "true", the absolute value of *semop* is added to the calling process's *semadj* value (see *exit(S)*) for the specified semaphore.

If *semval* is less than the absolute value of *semop* and (*sem\_flg* & IPC\_NOWAIT) is "true", *semop* will return immediately.



If *semval* is less than the absolute value of *semop* and (*sem\_flg* & **IPC\_NOWAIT**) is "false", *semop* will increment the *semncnt* associated with the specified semaphore and suspend execution of the calling process until one of the following conditions occurs:

*Semval* becomes greater than or equal to the absolute value of *semop*. When this occurs, the value of *semncnt* associated with the specified semaphore is decremented, the absolute value of *semop* is subtracted from *semval* and, if (*sem\_flg* & **SEM\_UNDO**) is "true", the absolute value of *semop* is added to the calling process's *semadj* value for the specified semaphore.

The *semid* for which the calling process is awaiting action is removed from the system (see *semctl(S)*). When this occurs, *errno* is set equal to **EIDRM**, and a value of -1 is returned.

The calling process receives a signal that is to be caught. When this occurs, the value of *semncnt* associated with the specified semaphore is decremented, and the calling process resumes execution in the manner prescribed in *signal(S)*.

If *semop* is a positive integer, the value of *semop* is added to *semval* and, if (*sem\_flg* & **SEM\_UNDO**) is "true", the value of *semop* is subtracted from the calling process's *semadj* value for the specified semaphore. {ALTER}

If *semop* is zero, one of the following will occur: {READ}

If *semval* is zero, *semop* will return immediately.

If *semval* is not equal to zero and (*sem\_flg* & **IPC\_NOWAIT**) is "true", *semop* will return immediately.

If *semval* is not equal to zero and (*sem\_flg* & **IPC\_NOWAIT**) is "false", *semop* will increment the *semzcnt* associated with the specified semaphore and suspend execution of the calling process until one of the following occurs:

*Semval* becomes zero, at which time the value of *semzcnt* associated with the specified semaphore is decremented.

The semid for which the calling process is awaiting action is removed from the system. When this occurs, *errno* is set equal to EIDRM, and a value of -1 is returned.

The calling process receives a signal that is to be caught. When this occurs, the value of *semzcnt* associated with the specified semaphore is decremented, and the calling process resumes execution in the manner prescribed in *signal(S)*.

The *semop* system call will fail if one or more of the following is true for any of the semaphore operations specified by *sops*:

- [EINVAL]        *semid* is not a valid semaphore identifier.
- [EFBIG]        *sem\_num* is less than zero or greater than or equal to the number of semaphores in the set associated with *semid*.
- [E2BIG]        *nsops* is greater than the system-imposed maximum.
- [EACCES]        Operation permission is denied to the calling process (see *intro(S)*).
- [EAGAIN]        The operation would result in suspension of the calling process but (*sem\_flg* & IPC\_NOWAIT) is "true".
- [ENOSPC]        The limit on the number of individual processes requesting an SEM\_UNDO would be exceeded.
- [EINVAL]        The number of individual semaphores for which the calling process requests an SEM\_UNDO would exceed the limit.
- [ERANGE]        An operation would cause a *semval* to overflow the system-imposed limit.
- [ERANGE]        An operation would cause a *semadj* value to overflow the system-imposed limit.
- [EFAULT]        *sops* points to an illegal address.

Upon successful completion, the value of *sempid* for each semaphore specified in the array pointed to by *sops* is set equal to the process ID of the calling process.

## See Also

---

exec(S), exit(S), fork(S), intro(S), semctl(S), semget(S)

## Diagnostics

---

If *semop* returns due to the receipt of a signal, a value of -1 is returned to the calling process and *errno* is set to EINTR. If it returns due to the removal of a *semid* from the system, a value of -1 is returned and *errno* is set to EIDRM.

Upon successful completion, a value of zero is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## Standards Conformance

---

*semop* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



## setbuf, setvbuf

---

assign buffering to a stream

### Syntax

---

```
#include <stdio.h>
```

```
void setbuf (stream, buf)
FILE *stream;
char *buf;
```

```
int setvbuf (stream, buf, type, size)
FILE *stream;
char *buf;
int type;
size_t size;
```

### Description

---

The *setbuf* function may be used after a stream has been opened but before it is read or written. It causes the array pointed to by *buf* to be used instead of an automatically allocated buffer. If *buf* is the NULL pointer, input/output will be completely unbuffered.

A constant **BUFSIZ**, defined in the `<stdio.h>` header file, tells how big an array is needed:

```
char buf[BUFSIZ];
```

*setvbuf* may be used after a stream has been opened but before it is read or written. *Type* determines how *stream* will be buffered. Legal values for *type* (defined in `stdio.h`) are:

`_IOFBF` causes input/output to be fully buffered.

`_IOLBF` causes output to be line buffered; the buffer will be flushed when a newline is written, the buffer is full, or input is requested.

`_IONBF` causes input/output to be completely unbuffered.

If *buf* is not the NULL pointer, the array it points to will be used for buffering, instead of an automatically allocated buffer. *size* specifies the size of the buffer to be used. The constant **BUFSIZ** in `<stdio.h>` is suggested as a good buffer size. If input/output is unbuffered, *buf* and *size* are ignored.

By default, output to a terminal is line-buffered and all other input/output is fully buffered.

## See Also

---

`fopen(S)`, `getc(S)`, `malloc(S)`, `putc(S)`, `stdio(S)`

## Diagnostics

---

If an illegal value for *type* or *size* is provided, *setvbuf* returns a non-zero value. Otherwise, the value returned will be zero.

## Notes

---

A common source of error is allocating buffer space as an “automatic” variable in a code block, and then failing to close the stream in the same block.

The parameter order of the *setvbuf* function is altered by the **-compat** option of the *cc*(CP) command.

## Standards Conformance

---

*setbuf* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
ANSI X3.159-198X C Language Draft Standard, May 13, 1988;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

*setvbuf* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
and ANSI X3.159-198X C Language Draft Standard, May 13, 1988.

## setgroups

---

set group access list

### Syntax

---

```
#include <sys/param.h>

setgroups(ngroups, gidset)
int gidsetsize;
gid_t *grouplist;
```

### Description

---

*setgroups* sets the group access list of the current user process according to the array *gidset*. The parameter *gidsetsize* indicates the number of entries in the array and must be no larger than the current value of NGROUPS. NGROUPS is a tunable kernel parameter.

Only the super-user may set new groups.

### Return Value

---

A 0 value is returned on success, -1 on error, with a error code stored in *errno*.

### Diagnostics

---

The *setgroups* call will fail if:

- |          |                                                                               |
|----------|-------------------------------------------------------------------------------|
| [EPERM]  | The caller is not the super-user.                                             |
| [EFAULT] | The address specified for <i>gidset</i> is outside the process address space. |
| [EINVAL] | The <i>gidsetsize</i> parameter is less than zero or greater than NGROUPS.    |

### See Also

---

getgroups(S)



# setjmp, longjmp

---

non-local goto

## Syntax

---

```
#include <setjmp.h>
```

```
int setjmp (env)
jmp_buf env;
```

```
void longjmp (env, val)
jmp_buf env;
int val;
```

## Description

---

These functions are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

*setjmp* saves its stack environment in *env* (whose type, *jmp\_buf*, is defined in the *<setjmp.h>* header file) for later use by *longjmp*. It returns the value 0.

*longjmp* restores the environment saved by the last call of *setjmp* with the corresponding *env* argument. After *longjmp* is completed, program execution continues as if the corresponding call of *setjmp* had just returned the value *val*. *longjmp* cannot cause *setjmp* to return the value 0. If *longjmp* is invoked with a second argument of 0, *setjmp* will return 1. At the time of the second return from *setjmp*, all external and static variables have values as of the time *longjmp* is called (see example). The values of register and automatic variables are undefined.

In a future release, C language users will be able to identify syntactically those automatic variables on whose values they need to rely after the second return from *setjmp*.

## Example

---

```
#include <setjmp.h>

jmp_buf env;
int i = 0;
main ()
{
 void exit ();

 if(setjmp(env) != 0) {
 (void) printf("value of i on 2nd return from setjmp: %d\n", i);
 exit(0);
 }
 (void) printf("value of i on 1st return from setjmp: %d\n", i);
 i = 1;
 g();
 /*NOTREACHED*/
}

g()
{
 longjmp(env, 1);
 /*NOTREACHED*/
}
```

If the **a.out** resulting from this C language code is run, the output will be:

```
value of i on 1st return from setjmp: 0
value of i on 2nd return from setjmp: 1
```

## See Also

---

signal(S)

## Warning

---

Problems may occur if *longjmp* is called before *env* is primed with a call to *setjmp*.

## Standards Conformance

---

*longjmp* and *setjmp* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
ANSI X3.159-198X C Language Draft Standard, May 13, 1988;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-  
Dependent System Support;  
and NIST FIPS 151-1.

## setlocale

---

set or read international environment

### Syntax

---

```
#include <locale.h>
```

```
char *setlocale(category, locale)
int category
char *locale
```

### Description

---

The function *setlocale* sets all or part of the current program's locale (definitions of language and conventions). Once invoked, *setlocale* immediately updates the information used by the routines that support locale-specific capabilities (for example character and string handling routines, collation, and so on).

The *setlocale* function accepts two arguments. The *category* argument describes the category of international information to set. The following categories (defined in **locale.h**) are recognised:

|                    |                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------|
| <b>LC_ALL</b>      | Set all categories in the international environment.                                          |
| <b>LC_COLLATE</b>  | Set only the category that contains collating information.                                    |
| <b>LC_CTYPE</b>    | Set only the category that contains character classification and case conversion information. |
| <b>LC_NUMERIC</b>  | Set only the category that contains numeric formatting information.                           |
| <b>LC_TIME</b>     | Set only the category that contains date and time formatting information.                     |
| <b>LC_MESSAGES</b> | Set only the category that contains message language information.                             |
| <b>LC_MONETARY</b> | Set only the category that contains currency formatting information.                          |

The *locale* argument should be a character string that corresponds to the name of a particular international environment. There are three special locale strings, as follows:



- "C" Defines the minimal environment needed for the C programming language.
- "" Specifies that the locale is set as defined in the user environment (see *environ*(M)).
- (char \*) 0 Used to query the current locale for a specific category. When returned by *setlocale*, indicates an error.

Other strings use the same format as the user environment variables (see *environ*(M)).

At program startup (before *main*), the equivalent of:

```
setlocale(LC_ALL, "");
```

is executed, so that the initial locale for the program is as defined in the user environment.

In addition to setting the locale, *setlocale* may be used to query the current locale. This is done by calling *setlocale* with a null pointer (i.e. (char \*)0) as the locale argument. *setlocale* will return the current locale associated with this category and the program's locale is not changed.

If *setlocale* is unable to successfully set the locale for any category, the active locale is unchanged; if the unsuccessful call was during program startup, the effect is as if the previous locale was "C". When the failing call was for the category **LC\_ALL**, all subcategories which could be set are changed; only those which could not be set remain as before.

## Return Value

---

If successful, *setlocale* returns the locale name string associated with the locale just set. On error, *setlocale* returns a null pointer.

When a null pointer is given as the locale, *setlocale* returns the locale name string of the current locale. If the individual categories differ at all, and the specified category is **LC\_ALL**, then an expanded locale name string is returned containing the information for each category.

The name string returned by *setlocale* is such that a subsequent call to *setlocale* with that locale name string and its associated category will restore that part of the program's locale.

When a *locale* is partially defined and does not have a corresponding entry in the file */etc/default/lang*, the following error message is displayed:

```
libc: setlocale: xxxx: no LANG= line in /etc/default/lang
```

where *xxxx* is one of the environment variables found in **environ** (M). For example, **LC\_CTYPE**.

When a *locale* is inaccurately defined, the following error message is displayed:

```
libc: setlocale: xxxx: cannot open locale file
```

where *xxxx* is one of the environment variables found in **environ** (M). For example, **LC\_CTYPE**.

## See Also

---

**environ**(M), **locale**(M), **ctype**(S), **collation**(S), **ctime**(S)

## Notes

---

The return values from *setlocale* are pointers to static data, the content of which is overwritten with each call.

## Standards Conformance

---

*setlocale* is conformant with:

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

## setluid

---

set login user ID

### Syntax

---

```
#include <sys/types.h>
#include <sys/security.h>
#include <sys/audit.h>
#include <prot.h>
```

```
int setluid (uid)
unsigned short uid;
```

### Description

---

*Setluid* is used to set the login user ID of the calling process. The login UID, or LUID, should be set at login time. Only the superuser can set the LUID. Once set, it cannot be reset, even by the superuser.

Until the LUID is set, the *setuid*(S) and *setgid*(S) system calls will fail. This ensures that the LUID is set before any identity changes in the other IDs.

The *setluid* call is expected to be invoked by the *login*(C) program just prior to the identity changes caused by *setuid*(S) and *setgid*(S) calls. It is also expected to be used by *at*(C) and *crontab*(C) job entries before starting a non-interactive session for a user.

The LUID is an accurate representation of the user who logged into the system and cannot be altered during the session. The LUID is needed because both the effective and real UIDs can be altered by use of *setuid*(S) and the setuid bits on an executable file, and at times during a session, will not accurately reflect the login user.

The LUID is inherited by all children of the process. If the LUID were not set before a *fork*(S), the child would also contain an unset LUID.

*Setluid* will fail if one or more of the following are true:

- [EPERM] The LUID has already been set for this process or any ancestors of this process.
- [EINVAL] *Uid* is out of range.



## Return Value

---

Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

## See Also

---

`getluid(S)`, `getuid(S)`, `setuid(S)`, `setgid(S)`, `stat(S)`

## Value Added

---

*setluid* is an extension of AT&T System V provided by the Santa Cruz Operation.

## setpgid

---

set process group ID for job control

### Syntax

---

```
#include <sys/types.h>
```

```
int setpgid (pid, pgid)
pid_t pid, pgid;
```

### Description

---

If {**POSIX\_JOB\_CONTROL**} is defined, the *setpgid()* function is used to either join an existing process group or create a new process group within the session of the calling process. The process group ID of a session leader does not change. Upon successful completion, the process group ID of the process with a process ID that matches *pid* is set to *pgid*. As a special case, if *pid* is zero, the process ID of the calling process is used. Also, if *pgid* is zero, the process ID of the indicated process is used.

Otherwise, either the implementation supports the *setpgid()* function as described above or the *setpgid()* fails.

### Return Value

---

Upon successful completion, the *setpgid()* function returns a value of zero. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

### Diagnostics

---

If any of the following conditions occur, the *setpgid()* function returns -1 and sets *errno* to the corresponding value:

[EACCES]

The value of the *pid* argument matches the process ID of a child process of the calling process and the child process

[EINVAL]

The value of the *pgid* argument is less than zero or is an unsupported value.

[ENOSYS]

The *setpgid()* function is not supported by this implementation.

**[EPERM]**

The process indicated by the *pid* argument is a session leader.

The value of the *pid* argument is valid but matches the process ID of a child process of the calling process and the child process is not in the same session as the calling process.

The value of the *pgid* argument does not match the process ID of the process indicated by the *pid* argument and there is no process with a process group ID that matches the value of the *pgid* argument in the same session as the calling process.

**[ESRCH]**

The value of the *pid* argument does not match the process ID of the calling process or of a child process of the calling process.

**See Also**

---

exec(S), getpid(S), setsid(S)

**Standards Conformance**

---

*setpgid* is conformant with:

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.



## **setpgrp**

---

set process group ID

### **Syntax**

---

**int setpgrp ( )**

### **Description**

---

The *setpgrp* system call sets the process group ID of the calling process to the process ID of the calling process and returns the new process group ID.

### **See Also**

---

*exec(S)*, *fork(S)*, *getpid(S)*, *intro(S)*, *kill(S)*, *signal(S)*

### **Diagnostics**

---

The *setpgrp* system call returns the value of the new process group ID.

### **Standards Conformance**

---

*setpgrp* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## setpriv

set system privileges for this process

### Syntax

```
#include <sys/types.h>
#include <sys/security.h>
#include <sys/audit.h>

int setpriv (privtype, privs)
int privtype;
priv_t *privs;
```

### Description

*setpriv* sets the system privilege vector for this process to that in the user-supplied *privs* vector. This vector should have at least **SEC\_SPRIVVEC\_SIZE** entries. The *privtype* argument may only contain the privilege type **SEC\_EFFECTIVE\_PRIV**.

At system initialization, all privileges are set. System privileges are inherited by all children of the process and must issue *setpriv* themselves to further restrict system privileges.

The system privilege vector contains per-process privileges used by the TCB. The following system privileges are defined:

- |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [SELFAUDIT] | The process does its own auditing. The system will not produce audit records for this process.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| [SUID]      | The process may execute SUID programs. Without this privilege, the process cannot execute any SUID program not set to the same process owner.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| [PROMAIN]   | Allow a SUID program to access any pathname, subject to the normal discretionary access checking. Without this privilege, a SUID program, after invoking <i>setuid</i> (S) to change identity from the program owner to the real user, may only access a pathname (restricted to the real user) in or under the current directory. Path names above the current directory are only accessible if the program owner may access them. Changing the current directory has no effect on this, for the current directory at the time of the SUID program execution (called the promain root) is remembered. Previously open files continue to be accessible, no matter how they were opened. Until this privilege was devised, a user had |

no protection against errant or malicious SUID programs. The privilege provides a means for the process to restrict the environment used by the SUID program, so that the program owner cannot usurp files owned by the real UID. With this privilege off, the user may run a SUID program with the current directory the root of a subtree that contains no important data therein. Any attempt to access a pathname above the current directory will return an error of [ENOENT]. This mechanism prevents many kinds of Trojan horses from SUID programs, where the SUID program uses the *setuid(S)* call to assign the effective UID to the real UID so that files inaccessible to the prior effective UID become accessible, all done without the knowledge or consent of the session user.

## [LABEL\_TERMINAL]

With this privilege, the process can output the string to set or change the terminal label, or otherwise modify the field where the terminal label resides. Without the privilege, the sequence to set the terminal label is intercepted by the system and altered to a harmless (to the label field) sequence.

## [SETID]

Allow a program to set the SUID or SGID bits on a file. Turning this privilege off prevents a new user from accidentally propagating his identity. Turning this privilege off and running an untrusted program prevents that program from secretly creating a file owned by you (like a copy of */bin/sh*) and setting the SUID bit so that it can run as you unrestricted. There are other similar uses.

## [SETOWNER]

Allow a program to give a file away (either the user or group). This privilege is needed for a user to execute the System V *chown(S)* call. Without this privilege, a user operates with the *chown* semantics of BSD, where a normal user cannot give a file away.

*setpriv* will fail if the following is true:

[EFAULT] *privs* points to an invalid address.

[EPERM] *privs* has more privileges set than the process currently has.

[EINVAL] *privtype* is not **SEC\_EFFECTIVE\_PRIV**.



## RETURN VALUE

---

Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## See Also

---

chdir(S), chroot(S), getpriv(S), audit(S)

## Value Added

---

*setpriv* is an extension of AT&T System V provided by the Santa Cruz Operation.

## setsid

---

create session and set process ID

### Syntax

---

```
#include <sys/types.h>
```

```
pid_t setsid ()
```

### Description

---

If the calling process is not a process group leader, the *setsid()* function creates a new session. The calling process is the session leader of this new session, the process group leader of a new process group, and has no controlling terminal. The process group ID of the calling process is set equal to the process ID of the calling process. The calling process is the only process in the new process group and the only process in the new session.

### Return Value

---

Upon successful completion, *setsid()* returns the value of the process group ID of the calling process. If any of the following conditions occur, the *setsid()* function shall return -1 and set *errno* to the corresponding value:

[EPERM]

The calling process is already a process group leader, or the process group ID of a process other than the calling process matches the process ID of the calling process.

### See Also

---

exec(S), exit(S), fork(S), getpid(S), kill(S), setpgid(S), sigaction(S)

### Standards Conformance

---

*setsid* is conformant with:

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

## setuid, setgid

---

### set user and group IDs

#### Syntax

---

```
int setuid (uid)
int uid;
```

```
int setgid (gid)
int gid;
```

#### Description

---

The *setuid* (*setgid*) system call is used to set the real user (group) ID and effective user (group) ID of the calling process.

If the effective user ID of the calling process is super-user, the real user (group) ID and effective user (group) ID are set to *uid* (*gid*).

If the effective user ID of the calling process is not super-user, but its real user (group) ID is equal to *uid* (*gid*), the effective user (group) ID is set to *uid* (*gid*).

If the effective user ID of the calling process is not super-user, but the saved set-user (group) ID from *exec*(S) is equal to *uid* (*gid*), the effective user (group) ID is set to *uid* (*gid*).

The *setuid* (*setgid*) system call will fail if the real user (group) ID of the calling process is not equal to *uid* (*gid*) and its effective user ID is not super-user. [EPERM]

The *uid* (*gid*) is out of range. [EINVAL]

#### See Also

---

getuid(S), intro(S)

#### Diagnostics

---

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.



## **Standards Conformance**

---

*setgid* and *setuid* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;

and NIST FIPS 151-1.

## shmctl

### shared memory control operations

---

#### Syntax

---

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmctl (shmid, cmd, buf)
int shmid, cmd;
struct shm_id *buf;
```

#### Description

---

The *shmctl* system call provides a variety of shared memory control operations as specified by *cmd*. The following *cmds* are available:

##### IPC\_STAT

Place the current value of each member of the data structure associated with *shmid* into the structure pointed to by *buf*. The contents of this structure are defined in *intro*(S). {READ}

##### IPC\_SET

Set the value of the following members of the data structure associated with *shmid* to the corresponding value found in the structure pointed to by *buf*:

```
shm_perm.uid
shm_perm.gid
shm_perm.mode /* only low 9 bits */
```

This *cmd* can only be executed by a process that has an effective user ID equal to that of super-user, or to the value of *shm\_perm.cuid* or *shm\_perm.uid* in the data structure associated with *shmid*.

##### IPC\_RMID

Remove the shared memory identifier specified by *shmid* from the system and destroy the shared memory segment and data structure associated with it. This *cmd* can only be executed by a process that has an effective user ID equal to that of super-user, or to the value of *shm\_perm.cuid* or *shm\_perm.uid* in the data structure associated with *shmid*.

**SHM\_LOCK**

Lock the shared memory segment specified by *shmid* in memory. This *cmd* can only be executed by a process that has an effective user ID equal to super-user.

**SHM\_UNLOCK**

Unlock the shared memory segment specified by *shmid*. This *cmd* can only be executed by a process that has an effective user ID equal to super-user.

The *shmctl* system call will fail if one or more of the following is true:

- [EINVAL] *shmid* is not a valid shared memory identifier.
- [EINVAL] *cmd* is not a valid command.
- [EACCES] *cmd* is equal to **IPC\_STAT** and {**READ**} operation permission is denied to the calling process (see *intro*(S)).
- [EPERM] *cmd* is equal to **IPC\_RMID** or **IPC\_SET**, and the effective user ID of the calling process is not equal to that of super-user or to the value of **shm\_perm.cuid** or **shm\_perm.uid** in the data structure associated with *shmid*.
- [EPERM] *cmd* is equal to **SHM\_LOCK** or **SHM\_UNLOCK**, and the effective user ID of the calling process is not equal to that of super-user.
- [EFAULT] *buf* points to an illegal address.
- [ENOMEM] *cmd* is equal to **SHM\_LOCK**, and there is not enough memory.

---

**See Also**

*shmget*(S), *shmop*(S)

---

**Diagnostics**

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

---

**Notes**

The user must explicitly remove shared memory segments after the last reference to them has been removed.



## **Standards Conformance**

---

*shmctl* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

# shmget

---

get shared memory segment identifier

## Syntax

---

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
```

```
int shmget (key, size, shmflg)
key_t key;
int size, shmflg;
```

## Description

---

The *shmget* system call returns the shared memory identifier associated with *key*.

A shared memory identifier and associated data structure and shared memory segment of at least *size* bytes (see *intro(S)*) are created for *key* if one of the following is true:

*key* is equal to **IPC\_PRIVATE**.

*key* does not already have a shared memory identifier associated with it, and (*shmflg* & **IPC\_CREAT**) is “true”.

Upon creation, the data structure associated with the new shared memory identifier is initialized as follows:

**shm\_perm.cuid**, **shm\_perm.uid**, **shm\_perm.cgid**, and **shm\_perm.gid** are set equal to the effective user ID and effective group ID, respectively, of the calling process.

The low-order 9 bits of **shm\_perm.mode** are set equal to the low-order 9 bits of *shmflg*. **shm\_segsz** is set equal to the value of *size*.

**shm\_lpid**, **shm\_nattch**, **shm\_atime**, and **shm\_dtime** are set equal to 0.

**shm\_ctime** is set equal to the current time.

The *shmget* system call will fail if one or more of the following is true:

[EINVAL] *size* is less than the system-imposed minimum or greater than the system-imposed maximum.

|          |                                                                                                                                                                                |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [EACCES] | A shared memory identifier exists for <i>key</i> , but operation permission (see <i>intro</i> (S)) as specified by the low-order 9 bits of <i>shmflg</i> would not be granted. |
| [EINVAL] | A shared memory identifier exists for <i>key</i> , but the size of the segment associated with it is less than <i>size</i> , and <i>size</i> is not equal to zero.             |
| [ENOENT] | A shared memory identifier does not exist for <i>key</i> , and ( <i>shmflg</i> & <i>IPC_CREAT</i> ) is "false".                                                                |
| [ENOSPC] | A shared memory identifier is to be created, but the system-imposed limit on the maximum number of allowed shared memory identifiers system wide would be exceeded.            |
| [ENOMEM] | A shared memory identifier and associated shared memory segment are to be created, but the amount of available memory is not sufficient to fill the request.                   |
| [EEXIST] | A shared memory identifier exists for <i>key</i> but ( <i>shmflg</i> & <i>IPC_CREAT</i> ) and ( <i>shmflg</i> & <i>IPC_EXCL</i> ) are "true".                                  |

## See Also

---

*intro*(S), *shmctl*(S), *shmop*(S)

## Diagnostics

---

Upon successful completion, a non-negative integer, namely a shared memory identifier is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## Notes

---

The user must explicitly remove shared memory segments after the last reference to them has been removed.

## Standards Conformance

---

*shmget* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



# shmop: shmat, shmdt

---

shared memory operations

## Syntax

---

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
```

```
char *shmat (shmidx, shmaddr, shmflg)
int shmidx;
char *shmaddr;
int shmflg;
```

```
int shmdt (shmaddr)
char *shmaddr;
```

## Description

---

The *shmat* system call attaches the shared memory segment associated with the shared memory identifier specified by *shmidx* to the data segment of the calling process. The segment is attached at the address specified by one of the following criteria:

If *shmaddr* is equal to zero, the segment is attached at the first available address as selected by the system.

If *shmaddr* is not equal to zero and (*shmflg* & **SHM\_RND**) is "true", the segment is attached at the address given by (*shmaddr* - (*shmaddr* modulus **SHMLBA**)).

If *shmaddr* is not equal to zero and (*shmflg* & **SHM\_RND**) is "false", the segment is attached at the address given by *shmaddr*.

*shmdt* detaches from the calling process's data segment the shared memory segment located at the address specified by *shmaddr*.

The segment is attached for reading if (*shmflg* & **SHM\_RDONLY**) is "true" {**READ**}; otherwise it is attached for reading and writing {**READ/WRITE**}.

*shmat* will fail and not attach the shared memory segment if one or more of the following is true:

[EINVAL]      *shmidx* is not a valid shared memory identifier.

|          |                                                                                                                                                     |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| [EACCES] | Operation permission is denied to the calling process (see <i>intro</i> (S)).                                                                       |
| [ENOMEM] | The available data space is not large enough to accommodate the shared memory segment.                                                              |
| [EINVAL] | <i>shmaddr</i> is not equal to zero, and the value of ( <i>shmaddr</i> - ( <i>shmaddr</i> modulus SHMLBA)) is an illegal address.                   |
| [EINVAL] | <i>shmaddr</i> is not equal to zero, ( <i>shmflg</i> & SHM_RND) is "false", and the value of <i>shmaddr</i> is an illegal address.                  |
| [EMFILE] | The number of shared memory segments attached to the calling process would exceed the system-imposed limit.                                         |
| [EINVAL] | <i>shmdt</i> will fail and not detach the shared memory segment if <i>shmaddr</i> is not the data segment start address of a shared memory segment. |

## See Also

---

exec(S), exit(S), fork(S), intro(S), shmctl(S), shmget(S)

## Diagnostics

---

Upon successful completion, the return value is as follows:

*shmat* returns the data segment start address of the attached shared memory segment.

*shmdt* returns a value of 0.

Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## Notes

---

The user must explicitly remove shared memory segments after the last reference to them has been removed.

## Standards Conformance

---

*shmat* and *shmdt* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## shutdown

---

flushes block I/O and halts the CPU

### Syntax

---

```
#include <sys/types.h>
#include <sys/param.h>
#include <sys/filsys.h>

void shutdown (sblk, nsblk, arg);
struct filsys *sblk, *nsblk;
int arg;
```

### Description

---

*shutdown* causes all information in memory that should be on disk to be written out. This includes modified super-blocks, modified inodes, and delayed block I/O. The super-blocks of all writable file systems are flagged 'clean', so that they can be remounted without cleaning when the system is rebooted. *shutdown* then prints "Normal System Shutdown" on the console and halts the CPU.

The system then stays down or reboots dependent on whether *arg* is 0 or 1.

If *sblk* is greater than 1, it specifies the address of a super-block to be written to the root device as the last I/O before the halt, provided that *nsblk* is given as its bit-wise inverse. This facility is provided to allow file system repair programs to supersede the system's copy of the root super-block with one of their own.

If *sblk* is 1, the second argument is a command and the third argument is the argument to the command. The CONFPANIC command, a system configurable system call, is given the argument 0 to stay down, or 1 to reboot. When *shutdown* is called in this way, the purpose is not to bring down the system, but rather, to give instructions to the kernel regarding the way to deal with the next panic.

*shutdown* locks out all other processes while it is doing its work. However, it is recommended that user processes be killed off (see *kill* (S)) before calling *shutdown* as some types of disk activity could cause file systems to not be flagged "clean".

The caller must be the super-user.



## See Also

---

fsck(ADM), haltsys(ADM), shutdown(ADM), mount(S), kill(S)

## Notes

---

This routine must be linked using the linker option **-lx**.

# sigaction

examine and change signal action

## Syntax

```
#include <signal.h>
```

```
int sigaction (sig, act, oact)
int sig;
struct sigaction *act, *oact;
```

## Description

The *sigaction()* function allows the calling process to examine or specify (or both) the action to be associated with a specific signal. The argument *sig* specifies the signal; acceptable values are defined in *<signal.h>*.

The structure *sigaction*, used to describe an action to be taken, is defined in the header *<signal.h>* to include at least the following members:

| Member Type        | Member Name       | Description                                                                           |
|--------------------|-------------------|---------------------------------------------------------------------------------------|
| <i>void (*)( )</i> | <i>sa_handler</i> | SIG_DFL, SIG_IGN, or pointer to a function.                                           |
| <i>sigset_t</i>    | <i>sa_mask</i>    | Additional set of signals to be blocked during execution of signal-catching function. |
| <i>int</i>         | <i>sa_flags</i>   | Special flags to affect behavior of signal.                                           |

If the argument *act* is not NULL, it points to a structure specifying the action to be associated with the specified signal. If the argument *oact* is not NULL, the action previously associated with the signal is stored in the location pointed to by the argument *oact*. If the argument *act* is NULL, signal handling is unchanged by this function call; thus, the call can be used to enquire about the current handling of a given signal. The *sa\_handler* field of the *sigaction* structure identifies the action to be associated with the specified signal. If the *sa\_handler* field specifies a signal-catching function, the *sa\_mask* field identifies a set of signals that are added to the process's signal mask before the signal-catching function is invoked. The SIGKILL and SIGSTOP signals are not be added to the signal mask using this mechanism; this restriction is enforced by the system without causing an error to be indicated.

The *sa\_flags* field can be used to modify the behavior of the specified signal.

The following flag bit, defined in the header `<signal.h>`, can be set in *sa\_flags*:

| Symbolic<br>Constant | Description                                |
|----------------------|--------------------------------------------|
| SA_NOCLDSTOP         | Do not generate SIGCHLD when children stop |

If *sig* is SIGCHLD and the SA\_NOCLDSTOP flag is not set in *sa\_flags*, and the implementation supports the SIGCHLD signal, a SIGCHLD signal is generated for the calling process whenever any of its child processes stop. If *sig* is SIGCHLD and the SA\_NOCLDSTOP flag is set in *sa\_flags*, the implementation does not generate a SIGCHLD signal in this way.

When a signal is caught by a signal-catching function installed by the *sigaction()* function, a new signal mask is calculated and installed for the duration of the signal-catching function (or until a call to either the *sigprocmask()* or *sigsuspend()* function is made). This mask is formed by taking the union of the current signal mask and the value of the *sa\_mask* for the signal being delivered, and then including the signal being delivered. If and when the user's signal handler returns normally, the original signal mask is restored.

Once an action is installed for a specific signal, it remains installed until another action is explicitly requested (by another call to the *sigaction()* function), or until one of the *exec* functions is called.

If the previous action for *sig* had been established by the *signal()* function, the values of the fields returned in the structure pointed to by *oact* are unspecified, and in particular *oact->su\_handler* is not necessarily the same value passed to the *signal()* function. However, if a pointer to the same structure or a copy thereof is passed to a subsequent call to the *sigaction()* function via the *act* argument, handling of the signal is as if the original call to the *signal()* function were repeated.

If the *sigaction()* function fails, no new signal handler is installed.

## Return Value

Upon successful completion a value of zero is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## See Also

kill(S), signal(S), sigset(S)



## Diagnostics

---

If any of the following conditions occur, the *sigaction()* function shall return -1 and set *errno* to the corresponding value:

- [EINVAL] The value of the *sig* argument is an invalid or unsupported signal number, or an attempt was made to catch a signal that cannot be caught or to ignore a signal that cannot be ignored. See *signal(S)*.

## Standards Conformance

---

*sigaction* is conformant with:

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

## **sigsetjmp, siglongjmp**

---

non-local jumps

### **Syntax**

---

```
#include <setjmp.h>
```

```
int sigsetjmp (env, savemask)
sigjmp_buf env;
int savemask;
```

```
void siglongjmp (env, val)
sigjmp_buf env;
int val;
```

### **Description**

---

The *sigsetjmp()* macro complies with the definition of the *setjmp()* function. If the value of the *savemask* argument is not zero, the *sigsetjmp()* function also saves the process's current signal mask (see *signal(S)*) as part of the calling environment.

The *siglongjmp()* macro complies with the definition of the *longjmp()* function. If and only if the *env* argument was initialized by a call to the *sigsetjmp()* function with a non-zero *savemask* argument, the *siglongjmp()* function restores the saved signal mask.

### **See Also**

---

*sigaction(S)*, *sigprocmask(S)*, *sigsuspend(S)*.

### **Standards Conformance**

---

*siglongjmp* and *sigsetjmp* are conformant with:

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

# signal

specify what to do upon receipt of a signal

## Syntax

```
#include <signal.h>
```

```
void (*signal (sig, func))()
```

```
int sig;
```

```
void (*func)();
```

## Description

The *signal* system call allows the calling process to choose one of three ways in which it is possible to handle the receipt of a specific signal. *sig* specifies the signal and *func* specifies the choice.

*sig* can be assigned any one of the following except **SIGKILL**:

|                |                   |                                             |
|----------------|-------------------|---------------------------------------------|
| <b>SIGHUP</b>  | 01                | hangup                                      |
| <b>SIGINT</b>  | 02                | interrupt                                   |
| <b>SIGQUIT</b> | 03 <sup>[1]</sup> | quit                                        |
| <b>SIGILL</b>  | 04 <sup>[1]</sup> | illegal instruction (not reset when caught) |
| <b>SIGTRAP</b> | 05 <sup>[1]</sup> | trace trap (not reset when caught)          |
| <b>SIGIOT</b>  | 06 <sup>[1]</sup> | IOT instruction                             |
| <b>SIGABRT</b> | 06                | used by abort, replaces SIGIOT              |
| <b>SIGEMT</b>  | 07 <sup>[1]</sup> | EMT instruction                             |
| <b>SIGFPE</b>  | 08 <sup>[1]</sup> | floating point exception                    |
| <b>SIGKILL</b> | 09                | kill (cannot be caught or ignored)          |
| <b>SIGBUS</b>  | 10 <sup>[1]</sup> | bus error                                   |
| <b>SIGSEGV</b> | 11 <sup>[1]</sup> | segmentation violation                      |
| <b>SIGSYS</b>  | 12 <sup>[1]</sup> | bad argument to system call                 |
| <b>SIGPIPE</b> | 13                | write on a pipe with no one to read it      |
| <b>SIGALRM</b> | 14                | alarm clock                                 |
| <b>SIGTERM</b> | 15                | software termination signal                 |
| <b>SIGUSR1</b> | 16                | user-defined signal 1                       |
| <b>SIGUSR2</b> | 17                | user-defined signal 2                       |
| <b>SIGCLD</b>  | 18 <sup>[2]</sup> | death of a child                            |
| <b>SIGPWR</b>  | 19 <sup>[2]</sup> | power fail                                  |
| <b>SIGPOLL</b> | 22 <sup>[3]</sup> | selectable event pending                    |

*func* is assigned one of three values: **SIG\_DFL**, **SIG\_IGN**, or a *function address*. **SIG\_DFL**, and **SIG\_IGN**, are defined in the include file `<signal.h>`. Each is a macro that expands to a constant expression of type pointer to function returning *void*, and has a unique value that matches no declarable function.



The actions prescribed by the values of *func* are as follows:

**SIG\_DFL**—terminate process upon receipt of a signal

Upon receipt of the signal *sig*, the receiving process is to be terminated with all of the consequences outlined in *exit*(S). See Note [1] below.

**SIG\_IGN**—ignore signal

The signal *sig* is to be ignored.

Note: the signal **SIGKILL** cannot be ignored.

*function address*—catch signal

Upon receipt of the signal *sig*, the receiving process is to execute the signal-catching function pointed to by *func*. The signal number *sig* will be passed as the only argument to the signal-catching function. Additional arguments are passed to the signal-catching function for hardware-generated signals. Before entering the signal-catching function, the value of *func* for the caught signal will be set to **SIG\_DFL** unless the signal is **SIGILL**, **SIGTRAP**, or **SIGPWR**.

Upon return from the signal-catching function, the receiving process will resume execution at the point it was interrupted.

When a signal that is to be caught occurs during a *read*(S), a *write*(S), an *open*(S), or an *ioctl*(S) system call on a slow device (like a terminal; but not a file), during a *pause*(S) system call, or during a *wait*(S) system call that does not return immediately due to the existence of a previously stopped or zombie process, the signal catching function will be executed. Then the interrupted system call may return a -1 to the calling process with *errno* set to **EINTR**.

The *signal* system call will not catch an invalid function argument, *func*, and results are undefined when an attempt is made to execute the function at the bad address.

Note: The signal **SIGKILL** cannot be caught.

A call to *signal* cancels a pending signal *sig* except for a pending **SIGKILL** signal.

The *signal* system call will fail if *sig* is an illegal signal number, including **SIGKILL**. [**EINVAL**]

## See Also

---

intro(S), kill(S), pause(S), ptrace(S), wait(S), setjmp(S), sigset(S), kill(C)

## Diagnostics

---

Upon successful completion, *signal* returns the previous value of *func* for the specified signal *sig*. Otherwise, a value of SIG\_ERR is returned and *errno* is set to indicate the error. SIG\_ERR is defined in the include file <signal.h>.

## Notes

---

- [1] If SIG\_DFL is assigned for these signals, in addition to the process being terminated, a “core image” will be constructed in the current working directory of the process, if the following conditions are met:

The effective user ID and the real user ID of the receiving process are equal.

An ordinary file named **core** exists and is writable or can be created. If the file must be created, it will have the following properties:

- a mode of 0666 modified by the file creation mask (see *umask*(S))
- a file owner ID that is the same as the effective user ID of the receiving process
- a file group ID that is the same as the effective group ID of the receiving process.

- [2] For the signals SIGCLD and SIGPWR, *func* is assigned one of three values: SIG\_DFL, SIG\_IGN, or a *function address*. The actions prescribed by these values are:

**SIG\_DFL**—ignore signal

The signal is to be ignored.

**SIG\_IGN**—ignore signal

The signal is to be ignored. Also, if *sig* is SIGCLD, the calling process's child processes will not create zombie processes when they terminate (see *exit*(S)).

*function address* —catch signal

If the signal is **SIGPWR**, the action to be taken is the same as that described above for *func* equal to *function address*. The same is true if the signal is **SIGCLD** with one exception: while the process is executing the signal-catching function, any received **SIGCLD** signals will be ignored. (This is the default action.) A *wait* call must proceed any call to *signal* that specifies **SIGCLD**.

In addition, **SIGCLD** affects the *wait* and *exit* system calls as follows:

- |             |                                                                                                                                                                                                                                                                       |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>wait</i> | If the <i>func</i> value of <b>SIGCLD</b> is set to <b>SIG_IGN</b> and a <i>wait</i> is executed, the <i>wait</i> will block until all of the calling process's child processes terminate; it will then return a value of -1 with <i>errno</i> set to <b>ECHILD</b> . |
| <i>exit</i> | If in the exiting process's parent process the <i>func</i> value of <b>SIGCLD</b> is set to <b>SIG_IGN</b> , the exiting process will not create a zombie process.                                                                                                    |

When processing a pipeline, the shell makes the last process in the pipeline the parent of the preceding processes. A process that may be piped into in this manner (and thus become the parent of other processes) should take care not to set **SIGCLD** to be caught.

- [3] **SIGPOLL** is issued when a file descriptor corresponding to a **STREAMS** (see *intro(S)*) file has a "selectable" event pending. A process must specifically request that this signal be sent using the **I\_SETSIG** *ioctl* call. Otherwise, the process will never receive **SIGPOLL**.

## Standards Conformance

---

*signal* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
 The X/Open Portability Guide II of January 1987;  
 and ANSI X3.159-198X C Language Draft Standard, May 13, 1988.



# **sigpending**

---

examine pending signals

## **Syntax**

---

```
#include <signal.h>
```

```
int sigpending (set)
sigset_t *set ;
```

## **Description**

---

The *sigpending()* function stores the set of signals that are blocked from delivery and pending for the calling process, in the space pointed to by the argument *set*.

## **Return Value**

---

Upon successful completion a value of zero is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## **See Also**

---

signal(S), sigprocmask(S), sigset(S)

## **Standards Conformance**

---

*sigpending* is conformant with:

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

## sigprocmask

---

examine and change blocked signals.

### Syntax

---

```
#include <signal.h>
```

```
int sigprocmask (how, set, oset)
int how;
sigset_t *set, *oset ;
```

### Description

---

The *sigprocmask()* function is used to examine or change (or both) the calling process's signal mask. If the value of the argument *set* is not NULL, it points to a set of signals to be used to change the currently blocked set.

The value of the argument *how* indicates the manner in which the set is changed, and consists of one of the following values, as defined in the header *<signal.h>*:

| Name        | Description                                                                                                                           |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------|
| SIG_BLOCK   | The resulting set is the union of the current set and the signal set pointed to by the argument <i>set</i> .                          |
| SIG_UNBLOCK | The resulting set is the intersection of the current set and the complement of the signal set pointed to by the argument <i>set</i> . |
| SIG_SETMASK | The resulting set is the signal set pointed to by the argument <i>set</i> .                                                           |

If the argument *oset* is not NULL, the previous mask is stored in the space pointed to by *oset*. If the value of the argument *set* is NULL, the value of the argument *how* is not significant and the process's signal mask is unchanged by this function call; thus, the call can be used to inquire about currently blocked signals.

If there are any pending unblocked signals after the call to the *sigprocmask()* function, at least one of those signals is delivered before the *sigprocmask()* function returns.

It is not possible to block the SIGKILL and SIGSTOP signals; this is enforced by the system without causing an error to be indicated.

If any of the SIGFPE , SIGILL , or SIGSEGV signals are generated while they are blocked, the result is undefined, unless the signal was generated by a call to the *kill()* function or the *raise()* function.

If the *sigprocmask()* function fails, the process's signal mask is not changed by this function call.

## Return Value

---

Upon successful completion a value of zero is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## See Also

---

*sigaction(S)*, *sigpending(S)*, *signal(S)*, *sigset(S)*, *sigsuspend(S)*

## Diagnostics

---

If any of the following conditions occur, the *sigprocmask()* function returns -1 and sets *errno* to the corresponding value:

[EINVAL]

The value of the *how* argument is not equal to one of the defined values.

## Standards Conformance

---

*sigprocmask* is conformant with:

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.



## **sigsem**

---

signals a process waiting on a semaphore

### **Syntax**

---

```
int sigsem(sem_num);
int sem_num;
```

### **Description**

---

*sigsem* signals a process that is waiting on the semaphore *sem\_num* that it may proceed and use the resource governed by the semaphore. *sigsem* is used in conjunction with *waitsem*(S) to allow synchronization of processes wishing to access a resource. One or more processes may *waitsem* on the given semaphore and will be put to sleep until the process which currently has access to the resource issues a *sigsem* call. If there are any waiting processes, *sigsem* causes the process which is next in line on the semaphore's queue to be rescheduled for execution. The semaphore's queue is organized in first in first out (FIFO) order.

### **See Also**

---

*creatsem*(S), *opensem*(S), *waitsem*(S)

### **System Compatibility**

---

*sigsem* can only be used to signal semaphores created under UNIX Version 3.0, not for UNIX System V semaphores.

### **Diagnostics**

---

*sigsem* returns the value (int) -1 if an error occurs. If *sem\_num* does not refer to a semaphore type file, *errno* is set to ENOTNAM. If *sem\_num* has not been previously opened by *opensem*, *errno* is set to EBADF. If the process issuing a *sigsem* call is not the current "owner" of the semaphore (i.e., if the process has not issued a *waitsem* call before the *sigsem*), *errno* is set to ENAVAIL.

### **Notes**

---

This feature is a XENIX specific enhancement and may not be present in all UNIX implementations. This function must be linked using the linker option **-lx**.

# sigset

---

## manipulate signal sets

### Syntax

---

```
#include <signal.h>

int sigemptyset (set)
sigset_t *set;

int sigfillset (set)
sigset_t *set;

int sigaddset (set)
sigset_t *set;
int signo;

int sigdelset (set)
sigset_t *set;
int signo;

int sigismember (set)
sigset_t *set
int signo;
```

### Description

---

The *sigsetops* primitives manipulate sets of signals. They operate on data objects addressable by the application, not on any set of signals known to the system, such as the set blocked from delivery to a process or the set pending for a process (see **signal(S)**.)

The *sigemptyset()* function initializes the signal set pointed to by the argument *set*, such that all signals are excluded.

The *sigfillset()* function initializes the signal set pointed to by the argument *set*, such that all signals are included.

Applications call either *sigemptyset()* or *sigfullset()* at least once for each object of type *sigset\_t* prior to any use of that object. If such an object is not initialized in this way, but is nonetheless supplied as an argument to any of the *sigaddset()*, *sigdelset()*, *sigismember()*, *sigaction()*, *sigprocmask()*, *sigpending()*, or *sigsuspend()* functions, the result is undefined.

The *sigaddset()* and *sigdelset()* functions respectively add and delete the individual signal specified by the value of the argument *signo* from the signal set pointed to by the argument *set*.

The *sigismember()* function tests whether the signal specified by the value of the argument *signo* is a member of the set pointed to by the argument *set*.

## Return Value

---

Upon successful completion, the *sigismember()* function retains a value of one if the specified signal is a member of the specified set, or a value of zero if it is not. Upon successful completion, the other functions return a value of zero. For all of the above functions, if an error is detected, a value of -1 is returned and *errno* is set to indicate the error.

## See Also

---

*sigaction(S)*, *signal(S)*

## Diagnostics

---

For each of the following conditions, if the condition is detected, the *sigaddset()*, *sigdelset()*, and *sigismember()* functions return a -1 and set *errno* to the corresponding value:

[EINVAL] The value of the signal argument is an invalid or unsupported signal number.

## Standards Conformance

---

*sigset* is conformant with:

AT&T SVID Issue 2, Select Code 307-127.



# sigsuspend

---

wait for signal

## Syntax

---

```
#include <signal.h>
```

```
int sigsuspend (sigmask)
sigset_t *sigmask ;
```

## Description

---

The *sigsuspend()* function replaces the process's signal mask with the set of signals pointed to by the argument *sigmask* and then suspends the process until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process.

If the action is to terminate the process, the *sigsuspend()* function does not return. If the action is to execute a signal-catching function, the *sigsuspend()* returns after the signal-catching function returns, with the signal mask restored to the set that existed prior to the *sigsuspend()* call.

It is not possible to block those signals that cannot be ignored (see *signal(S)*); this is enforced by the system without causing an error to be indicated.

## Return Value

---

Since the *sigsuspend()* function suspends process execution indefinitely, there is no successful completion value returned. A value of -1 is returned and *errno* is set to indicate the error.

## Diagnostics

---

If any of the following conditions occur, the *sigsuspend()* function returns -1 and sets *errno* to the corresponding value:

[EINTR]

A signal is caught by the calling process and control is returned from the signal-catching function.

## See Also

---

pause(S), sigaction(S), signal(S), sigpending(S), sigprocmask(S),  
sigset(S)

## Standards Conformance

---

*sigsuspend* is conformant with:

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;  
and NIST FIPS 151-1.

# **sinh, cosh, tanh**

---

hyperbolic functions

## **Syntax**

---

```
#include <math.h>
```

```
double sinh (x)
double x;
```

```
double cosh (x)
double x;
```

```
double tanh (x)
double x;
```

## **Description**

---

The *sinh*, *cosh*, and *tanh* functions return, respectively, the hyperbolic sine, cosine and tangent of their argument.

## **See Also**

---

`matherr(S)`

## **Diagnostics**

---

The *sinh* and *cosh* functions return **HUGE** (and *sinh* may return **-HUGE** for negative *x*) when the correct value would overflow and set *errno* to **ERANGE**.

These error-handling procedures may be changed with the function *matherr(S)*.

## **Standards Conformance**

---

*cosh*, *sinh* and *tanh* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.



## sleep

---

suspend execution for interval

### Syntax

---

unsigned sleep (seconds)  
unsigned seconds;

### Description

---

The current process is suspended from execution for the number of *seconds* specified by the argument. The actual suspension time may be less than that requested for two reasons: (1) Because scheduled wakeups occur at fixed 1-second intervals, (on the second, according to an internal clock) and (2) because any caught signal will terminate the *sleep* following execution of that signal's catching routine. Also, the suspension time may be longer than requested by an arbitrary amount due to the scheduling of other activity in the system. The value returned by *sleep* will be the "unslept" amount (the requested time minus the time actually slept) in case the caller had an alarm set to go off earlier than the end of the requested *sleep* time, or premature arousal due to another caught signal.

The routine is implemented by setting an alarm signal and pausing until it (or some other signal) occurs. The previous state of the alarm signal is saved and restored. The calling program may have set up an alarm signal before calling *sleep*. If the *sleep* time exceeds the time till such alarm signal, the process sleeps only until the alarm signal would have occurred. The caller's alarm catch routine is executed just before the *sleep* routine returns. But if the *sleep* time is less than the time till such alarm, the prior alarm time is reset to go off at the same time it would have without the intervening *sleep*.

### See Also

---

alarm(S), pause(S), signal(S)

### Standards Conformance

---

*sleep* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

## **sputl, sgetl**

---

access long integer data in a machine-independent fashion

### **Syntax**

---

```
void sputl (value, buffer)
long value;
char *buffer;
```

```
long sgetl (buffer)
char *buffer;
```

### **Description**

---

*sputl* takes the four bytes of the long integer *value* and places them in memory starting at the address pointed to by *buffer*. The ordering of the bytes is the same across all machines.

*sgetl* retrieves the four bytes in memory starting at the address pointed to by *buffer* and returns the long integer value in the byte ordering of the host machine.

The combination of *sputl* and *sgetl* provides a machine-independent way of storing long numeric data in a file in binary form without conversion to characters.

A program that uses these functions must be linked with the **-lld** flag.

### **Standards Conformance**

---

*sgetl* and *sputl* are conformant with:

AT&T SVID Issue 2, Select Code 307-127.

# ssignal, gsignal

---

## software signals

### Syntax

---

```
#include <signal.h>
```

```
int (*ssignal (sig, action)) ()
int sig, (*action) ();
```

```
int gsignal (sig)
int sig;
```

### Description

---

The *ssignal* and *gsignal* functions implement a software facility similar to *signal*(S). This facility is used by the Standard C Library to enable users to indicate the disposition of error conditions, and is also made available to users for their own purposes.

Software signals made available to users are associated with integers in the inclusive range 1 through 16. A call to *ssignal* associates a procedure, *action*, with the software signal *sig*; the software signal, *sig*, is raised by a call to *gsignal*. Raising a software signal causes the action established for that signal to be *taken*.

The first argument to *ssignal* is a number identifying the type of signal for which an action is to be established. The second argument defines the action; it is either the name of a (user-defined) *action function* or one of the manifest constants **SIG\_DFL** (default) or **SIG\_IGN** (ignore). The *ssignal* function returns the action previously established for that signal type; if no action has been established or the signal number is illegal, *ssignal* returns **SIG\_DFL**.

The *gsignal* function raises the signal identified by its argument, *sig*:

If an action function has been established for *sig*, then that action is reset to **SIG\_DFL** and the action function is entered with argument *sig*. *gsignal* returns the value returned to it by the action function.

If the action for *sig* is **SIG\_IGN**, *gsignal* returns the value 1 and takes no other action.

If the action for *sig* is **SIG\_DFL**, *gsignal* returns the value 0 and takes no other action.



If *sig* has an illegal value or no action was ever specified for *sig*, *gsignal* returns the value 0 and takes no other action.

## See Also

---

signal(S), sigset(S)

## Notes

---

There are some additional signals with numbers outside the range 1 through 16 which are used by the Standard C Library to indicate error conditions. Thus, some signal numbers outside the range 1 through 16 are legal, although their use may interfere with the operation of the Standard C Library.

## Standards Conformance

---

*gsignal* and *ssignal* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## stat, fstat

---

get file status

### Syntax

---

```
#include <sys/types.h>
#include <sys/stat.h>
```

```
int stat (path, buf)
char *path;
struct stat *buf;
```

```
int fstat (fildes, buf)
int fildes;
struct stat *buf;
```

### Description

---

*path* points to a path name naming a file. Read, write, or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be searchable. The *stat* system call obtains information about the named file.

Note that in a Remote File Sharing environment, the information returned by *stat* depends upon the user/group mapping set-up between the local and remote computers (see *idload*(ADM)).

*fstat* obtains information about an open file known by the file descriptor *fildes*, obtained from a successful *open*, *creat*, *dup*, *fcntl*, or *pipe* system call.

*buf* is a pointer to a *stat* structure into which information is placed concerning the file.

The contents of the structure pointed to by *buf* include the following members:

|                     |                                                                                                                                                                                                                                                 |                                                                                                                                                                           |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>ushort</code> | <code>st_mode;</code>                                                                                                                                                                                                                           | <code>/* File mode [see <i>mknod</i>(S)] */</code>                                                                                                                        |
| <code>ino_t</code>  | <code>st_ino;</code>                                                                                                                                                                                                                            | <code>/* Inode number */</code>                                                                                                                                           |
| <code>dev_t</code>  | <code>st_dev;</code>                                                                                                                                                                                                                            | <code>/* ID of device containing */</code><br><code>/* a directory entry for this file */</code>                                                                          |
| <code>dev_t</code>  | <code>st_rdev;</code>                                                                                                                                                                                                                           | <code>/* ID of device */</code><br><code>/* This entry is defined only for */</code><br><code>/* character special or */</code><br><code>/* block special files */</code> |
| <code>short</code>  | <code>st_nlink;</code>                                                                                                                                                                                                                          | <code>/* Number of links */</code>                                                                                                                                        |
| <code>ushort</code> | <code>st_uid;</code>                                                                                                                                                                                                                            | <code>/* User ID of the file's owner */</code>                                                                                                                            |
| <code>ushort</code> | <code>st_gid;</code>                                                                                                                                                                                                                            | <code>/* Group ID of the file's group */</code>                                                                                                                           |
| <code>off_t</code>  | <code>st_size;</code>                                                                                                                                                                                                                           | <code>/* File size in bytes */</code>                                                                                                                                     |
| <code>time_t</code> | <code>st_atime;</code>                                                                                                                                                                                                                          | <code>/* Time of last access */</code>                                                                                                                                    |
| <code>time_t</code> | <code>st_mtime;</code>                                                                                                                                                                                                                          | <code>/* Time of last data modification */</code>                                                                                                                         |
| <code>time_t</code> | <code>st_ctime;</code>                                                                                                                                                                                                                          | <code>/* Time of last file status change */</code><br><code>/* Times measured in seconds since */</code><br><code>/* 00:00:00 GMT, Jan. 1, 1970 */</code>                 |
| <b>st_mode</b>      | The mode of the file as described in the <i>mknod</i> (S) system call.                                                                                                                                                                          |                                                                                                                                                                           |
| <b>st_ino</b>       | This field uniquely identifies the file in a given file system. The pair <i>st_ino</i> and <i>st_dev</i> uniquely identifies regular files.                                                                                                     |                                                                                                                                                                           |
| <b>st_dev</b>       | This field uniquely identifies the file system that contains the file. Its value may be used as input to the <i>ustat</i> (S) system call to determine more information about this file system. No other meaning is associated with this value. |                                                                                                                                                                           |
| <b>st_rdev</b>      | This field should be used only by administrative commands. It is valid only for block special or character special files and only has meaning on the system where the file was configured.                                                      |                                                                                                                                                                           |
| <b>st_nlink</b>     | This field should be used only by administrative commands.                                                                                                                                                                                      |                                                                                                                                                                           |
| <b>st_uid</b>       | The user ID of the file's owner.                                                                                                                                                                                                                |                                                                                                                                                                           |
| <b>st_gid</b>       | The group ID of the file's group.                                                                                                                                                                                                               |                                                                                                                                                                           |



|                 |                                                                                                                                                                                                                                                     |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>st_size</b>  | For regular files, this is the address of the end of the file. For pipes or fifos, this is the count of the data currently in the file. For block special or character special, this is not defined.                                                |
| <b>st_atime</b> | Time when file data was last accessed. Changed by the following system calls: <i>creat</i> (S), <i>mknod</i> (S), <i>pipe</i> (S), <i>utime</i> (S), and <i>read</i> (S).                                                                           |
| <b>st_mtime</b> | Time when data was last modified. Changed by the following system calls: <i>creat</i> (S), <i>mknod</i> (S), <i>pipe</i> (S), <i>utime</i> (S), and <i>write</i> (S).                                                                               |
| <b>st_ctime</b> | Time when file status was last changed. Changed by the following system calls: <i>chmod</i> (S), <i>chown</i> (S), <i>creat</i> (S), <i>link</i> (S), <i>mknod</i> (S), <i>pipe</i> (S), <i>unlink</i> (S), <i>utime</i> (S), and <i>write</i> (S). |

The *stat* system call will fail if one or more of the following is true:

|             |                                                                                          |
|-------------|------------------------------------------------------------------------------------------|
| [ENOTDIR]   | A component of the path prefix is not a directory.                                       |
| [ENOENT]    | The named file does not exist.                                                           |
| [EACCES]    | Search permission is denied for a component of the path prefix.                          |
| [EFAULT]    | <i>buf</i> or <i>path</i> points to an invalid address.                                  |
| [EINTR]     | A signal was caught during the <i>stat</i> system call.                                  |
| [ENOLINK]   | <i>path</i> points to a remote machine and the link to that machine is no longer active. |
| [EMULTIHOP] | Components of <i>path</i> require hopping to multiple remote machines.                   |

*Fstat* will fail if one or more of the following is true:

|           |                                                                                            |
|-----------|--------------------------------------------------------------------------------------------|
| [EBADF]   | <i>fildes</i> is not a valid open file descriptor.                                         |
| [EFAULT]  | <i>buf</i> points to an invalid address.                                                   |
| [ENOLINK] | <i>fildes</i> points to a remote machine and the link to that machine is no longer active. |

## See Also

---

chmod(S), chown(S), creat(S), link(S), mknod(S), pipe(S), read(S), time(S), unlink(S), utime(S), write(S)

## Diagnostics

---

Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## Standards Conformance

---

*fstat* and *stat* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

## statfs, fstatfs

get file system information

### Syntax

```
#include <sys/types.h>
#include <sys/statfs.h>

int statfs (path, buf, len, fstyp)
char *path;
struct statfs *buf;
int len, fstyp;

int fstatfs (fildes, buf, len, fstyp)
int fildes;
struct statfs *buf;
int len, fstyp;
```

### Description

The *statfs* system call returns a “generic superblock” describing a file system. It can be used to acquire information about mounted as well as unmounted file systems, and usage is slightly different in the two cases. In all cases, *buf* is a pointer to a structure (described below) which will be filled by the system call, and *len* is the number of bytes of information which the system should return in the structure. *len* must be no greater than **sizeof (struct statfs)** and ordinarily it will contain exactly that value; if it holds a smaller value, the system will fill the structure with that number of bytes. (This allows future versions of the system to grow the structure without invalidating older binary programs.)

If the file system of interest is currently mounted, *path* should name a file which resides on that file system. In this case the file system type is known to the operating system and the *fstyp* argument must be zero. For an unmounted file system *path* must name the block special file containing it and *fstyp* must contain the (non-zero) file system type. In both cases read, write, or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be searchable.

The *statfs* structure pointed to by *buf* includes the following members:

|       |           |                              |
|-------|-----------|------------------------------|
| short | f_fstyp;  | /* File system type */       |
| short | f_bsize;  | /* Block size */             |
| short | f_fsize;  | /* Fragment size */          |
| long  | f_blocks; | /* Total number of blocks */ |
| long  | f_bfree;  | /* Count of free blocks */   |



```

long f_files; /* Total number of file nodes */
long f_ffree; /* Count of free file nodes */
char f_fname[6]; /* Volume name */
char f_fpack[6]; /* Pack name */

```

The *fstatfs* system call is similar, except that the file named by *path* in *statfs* is instead identified by an open file descriptor *fildev* obtained from a successful *open*(S), *creat*(S), *dup*(S), *fcntl*(S), or *pipe*(S) system call.

The *statfs* system call obsoletes *ustat*(S) and should be used in preference to it in new programs.

The *statfs* and *fstatfs* system calls will fail if one or more of the following is true:

- [ENOTDIR]        A component of the path prefix is not a directory.
- [ENOENT]        The named file does not exist.
- [EACCES]        Search permission is denied for a component of the path prefix.
- [EFAULT]        *buf* or *path* points to an invalid address.
- [EBADF]        *fildev* is not a valid open file descriptor.
- [EINVAL]        *fstyp* is an invalid file system type; *path* is not a block special file and *fstyp* is nonzero; *len* is negative or is greater than **sizeof (struct statfs)**.
- [ENOLINK]       *path* points to a remote machine, and the link to that machine is no longer active.
- [EMULTIHOP]     Components of *path* require hopping to multiple remote machines.

## See Also

---

*chmod*(S), *chown*(S), *creat*(S), *link*(S), *mknod*(S), *pipe*(S), *read*(S), *time*(S), *unlink*(S), *utime*(S), *write*(S), *fs*(F)

## Diagnostics

---

Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## stdio

### standard buffered input/output package

---

### Syntax

---

**#include** <stdio.h>

**FILE** \*stdin, \*stdout, \*stderr;

### Description

---

The functions described in this manual entry constitute an efficient, user-level I/O buffering scheme. The in-line macros *getc*(S) and *putc*(S) handle characters quickly. The macros *getchar* and *putchar*, and the higher-level routines *fgetc*, *fgets*, *fprintf*, *fputc*, *fputs*, *fread*, *fscanf*, *fwrite*, *gets*, *getw*, *printf*, *puts*, *putw*, and *scanf* all use or act as if they use *getc* and *putc*; they can be freely intermixed.

A file with associated buffering is called a *stream* and is declared to be a pointer to a defined type **FILE**. The *fopen*(S) function creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. Normally, there are three open streams with constant pointers declared in the <stdio.h> header file and associated with the standard open files:

|               |                      |
|---------------|----------------------|
| <b>stdin</b>  | standard input file  |
| <b>stdout</b> | standard output file |
| <b>stderr</b> | standard error file  |

A constant **NULL** (0) designates a nonexistent pointer.

An integer-constant **EOF** (-1) is returned upon end-of-file or error by most integer functions that deal with streams (see the individual descriptions for details).

An integer constant **BUFSIZ** specifies the size of the buffers used by the particular implementation.

Any program that uses this package must include the header file of pertinent macro definitions, as follows:

```
#include <stdio.h>
```

The functions and constants mentioned in the entries of sub-class 3S of this manual are declared in that header file and need no further declaration. The constants and the following "functions" are implemented as macros (redeclaration of these names is perilous): *getc*, *getchar*, *putc*, *putchar*, *ferror*, *feof*, *clearerr*, and *fileno*.

Output streams, with the exception of the standard error stream *stderr*, are by default buffered if the output refers to a file, and line-buffered if the output refers to a terminal. The standard error output stream *stderr* is by default unbuffered, but use of *freopen* (see *fopen*(S)) will cause it to become buffered or line-buffered. When an output stream is unbuffered, information is queued for writing on the destination file or terminal as soon as written. When it is buffered, many characters are saved up and written as a block. When it is line-buffered, each line of output is queued for writing on the destination terminal as soon as the line is completed (that is, as soon as a new-line character is written or terminal input is requested). The *setbuf*(S) or *setvbuf*() functions in *setbuf*(S) may be used to change the stream's buffering strategy.

## See Also

---

*open*(S), *close*(S), *lseek*(S), *pipe*(S), *read*(S), *write*(S), *ctermid*(S), *cuserid*(S), *fclose*(S), *ferror*(S), *fopen*(S), *fread*(S), *fseek*(S), *getc*(S), *gets*(S), *popen*(S), *printf*(S), *putc*(S), *puts*(S), *scanf*(S), *setbuf*(S), *system*(S), *tmpfile*(S), *tmpnam*(S), *ungetc*(S)

## Diagnostics

---

Invalid *stream* pointers will usually cause grave disorder, possibly including program termination. Individual function descriptions describe the possible error conditions.

## Standards Conformance

---

*stdio* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



## stdipc: ftok

---

standard interprocess communication package

### Syntax

---

```
#include <sys/types.h>
#include <sys/ipc.h>
```

```
key_t ftok(path, id)
char *path;
char id;
```

### Description

---

All interprocess communication facilities require the user to supply a key to be used by the *msgget*(S), *semget*(S), and *shmget*(S) system calls to obtain interprocess communication identifiers. One suggested method for forming a key is to use the *ftok* subroutine described below. Another way to compose keys is to include the project ID in the most significant byte and to use the remaining portion as a sequence number. There are many other ways to form keys, but it is necessary for each system to define standards for forming them. If some standard is not adhered to, it will be possible for unrelated processes to unintentionally interfere with each other's operation. Therefore, it is strongly suggested that the most significant byte of a key in some sense refer to a project so that keys do not conflict across a given system.

*ftok* returns a key based on *path* and *id* that is usable in subsequent *msgget*, *semget*, and *shmget* system calls. *path* must be the path name of an existing file that is accessible to the process. *id* is a character which uniquely identifies a project. Note that *ftok* will return the same key for linked files when called with the same *id*, and that it will return different keys when called with the same file name but different *ids*.

### See Also

---

*intro*(S), *msgget*(S), *semget*(S), *shmget*(S)

### Diagnostics

---

*ftok* returns (**key\_t**) -1 if *path* does not exist or if it is not accessible to the process.

## Warning

---

If the file whose *path* is passed to *flok* is removed when keys still refer to the file, future calls to *flok* with the same *path* and *id* will return an error. If the same file is recreated, then *flok* is likely to return a different key than it did the original time it was called.

## stime

---

set time

### Syntax

---

```
int stime (tp)
long *tp;
```

### Description

---

The *stime* system call sets the system's idea of the time and date. *tp* points to the value of time as measured in seconds from 00:00:00 GMT January 1, 1970.

[EPERM]            *stime* will fail if the effective user ID of the calling process is not super-user.

### See Also

---

time(S)

### Diagnostics

---

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

### Standards Conformance

---

*stime* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



## stopio

---

stop further I/O to an open file

### Syntax

---

```
#include <sys/types.h>
#include <sys/security.h>
#include <sys/audit.h>
#include <prot.h>
```

```
int stopio (path)
char *path;
```

### Description

---

*Stopio* prevents further I/O on all file descriptors now having *path* open. The next time a *read*(S), *write*(S) or *ioctl*(S) is invoked by any process on any file descriptor of *path*, the system call will fail with the [EBADF] error and the SIGSYS signal is posted to the process. *Stopio* may only be invoked by the superuser or owner of file *path*.

*Stopio* is used to isolate successive user sessions on a single terminal. At the time *stopio* is invoked with the terminal *path* as the argument, background processes left by previous sessions may continue to run on the terminal, but any further input or output to the terminal will fail as described above. As the terminal gets re-opened after the *stopio* invocation, I/O using those new file descriptors will succeed until the next *stopio* call on the same *path*. This system call is intended to be used by a program like *getty*(1M) immediately before the open of the terminal line.

*Stopio* will fail if one of the following is true:

- |           |                                                                                                         |
|-----------|---------------------------------------------------------------------------------------------------------|
| [ENOTDIR] | A component of the path prefix is not a directory.                                                      |
| [ENOENT]  | The named file does not exist.                                                                          |
| [EACCES]  | Search permission is denied on a component of the path prefix.                                          |
| [EPERM]   | The effective user ID does not match the owner of the file and the effective user ID is not super-user. |
| [EROFS]   | The named file resides on a read-only file system. (This may go away - see Notes below.)                |

- [EFAULT] Path points outside the allocated address space of the process.
- [ENOTTY] *Path* is not a local character special file. (This may go away - see Notes below.)

## Return Value

---

Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## See Also

---

*open(S)*, *creat(S)*, *read(S)*, *write(S)*, *ioctl(S)*

## Notes

---

The [EROFS] error is not really appropriate for this system call.

*Stopio* should work for all file types, but a security study has not yet been made as to the utility or feasibility in such an assignment. For now, it works only for character special files to specifically address the problem of sharing a single terminal through multiple login sessions.

This functionality should really be a command within *fcntl(S)*, but *fcntl* expects the file to be open already and this call requires a path name.

## Value Added

---

*stopio* is an extension of AT&T System V provided by the Santa Cruz Operation.

## Example

---

```
#include <string.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

extern int errno;
int errnum;
int fh1, fh2;

main()
{
 errnum=0;
 if ((fh1=open("xxxx",O_RDONLY)) == -1)
 errnum=errno;
 fh2=open("yyyy",O_RDONLY);

 /* Other code that may set the errno value.*/
 if (errnum != 0)
 printf(streerror(errnum));
}
```

The program shown above tries to open files *xxxx* and *yyyy*. If an error occurs opening *xxxx*, the variable *errnum* is set to the *errno* value returned by *open*. Other code that may alter the *errno* value is then executed. Later, the saved *errno* value in *errnum* is checked and, if nonzero, an error message assigned to it by *streerror* is printed. If file *xxxx* does not exist, the example will print the following message:

No such file or directory

## Standards Conformance

---

*streerror* is conformant with:

ANSI X3.159-198X C Language Draft Standard, May 13, 1988.



## strerror

---

gets error message pointer from last routine call error

### Syntax

---

```
#include <string.h>
```

```
char *strerror(errnum)
int errnum;
```

### Description

---

The *strerror* routine maps *errnum* to an error-message string, returning a pointer to the string. The function itself does not actually print the message; for that, you need to call an output function such as *printf*.

### Return Value

---

The *strerror* function returns a pointer to the error-message string. The string can be overwritten by subsequent calls to *strerror*.

### See Also

---

clearerr(S), ferror(S), perror(S)

## strftime

---

format date/time string

### Syntax

---

```
#include <time.h>
```

```
int strftime(s, maxsize, format, timeptr)
char *s;
int maxsize;
char *format;
struct tm *timeptr;
```

### Description

---

The *strftime* function places its output (according to the format string *format* and the time values contained in the structure pointed to by *timeptr*), followed by the null character (\0) in consecutive bytes starting at *s*. The maximum length of the output string is specified by *maxsize*, the maximum number of characters including the terminating null character.

The *strftime* function converts and formats the time values contained in the structure pointed to by *timeptr* under control of the format string *format*. The string *format* is a character string can contain two types of object. These are: plain characters, which are simply copied to the output string, and directives.

Each directive is introduced by the percent character (%). The following directives are independent of the program's current locale (see *locale*(M)):

- %d** is replaced by the day of the month as a decimal number (01-31)
- %H** is replaced by the hour (24-hour clock) as a decimal number (00-23)
- %I** is replaced by the hour (12-hour clock) as a decimal number (01-12)
- %j** is replaced by the day of the year as a decimal number (001-366)
- %m** is replaced by the month as a decimal number (01-12)

- %M** is replaced by the minute as a decimal number (00-59)
- %S** is replaced by the second as a decimal number (00-59)
- %U** is replaced by the week number of the year (Sunday as the first day of the week) as a decimal number (00-52)
- %w** is replaced by the weekday as a decimal number [0(Sunday)-6]
- %W** is replaced by the week number of the year (Monday as the first day of the week) as a decimal number (00-52)
- %y** is replaced by the year without century as a decimal number (00-99)
- %Y** is replaced by the year with century as a decimal number
- %Z** is replaced by the timezone name, or by no characters if no timezone exists
- %%** is replaced by %

The strings used to replace the following directives are dependent on the program's current locale, and are defined using the utility *timtbl*(M):

- %a** is replaced by the locale's abbreviated weekday name
- %A** is replaced by the locale's full weekday name
- %b** is replaced by the locale's abbreviated month name
- %B** is replaced by the locale's full month name
- %c** is replaced by the locale's appropriate date and time representation
- %p** is replaced by the locale's equivalent of AM or PM
- %x** is replaced by the locale's appropriate date representation
- %X** is replaced by the locale's appropriate time representation

## Return Value

---

The function returns the number of characters placed in the string *s* (not including the terminating null character), or zero if the length of the output string would exceed *maxsize*. In the latter case, the content of the output string *s* is undefined.



Zero is also returned if the function detects a situation which would lead to infinite recursion; for example, if a format string refers to itself.

## See Also

---

timtbl(M), ctime(S), nl\_cxtime(S)

## Standards Conformance

---

*strftime* is conformant with:

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;  
and NIST FIPS 151-1.

**string: strcat, strdup, strncat,  
strcmp, strncmp, strcpy, strncpy,  
strlen, strchr, strrchr, strpbrk, strspn,  
strcspn, strtok**

---

string operations

## Syntax

---

```
#include <string.h>
#include <sys/types.h>
```

```
char *strcat (s1, s2)
const char *s1, *s2;
```

```
char *strdup (s1)
const char *s1;
```

```
char *strncat (s1, s2, n)
const char *s1, *s2;
size_t n;
```

```
int strcmp (s1, s2)
const char *s1, *s2;
```

```
int strncmp (s1, s2, n)
const char *s1, *s2;
size_t n;
```

```
char *strcpy (s1, s2)
const char *s1, *s2;
```

```
char *strncpy (s1, s2, n)
const char *s1, *s2;
size_t n;
```

```
size_t strlen (s)
const char *s;
```

```
char *strchr (s, c)
const char *s;
int c;
```

```
char *strrchr (s, c)
const char *s;
int c;
```

```
char *strpbrk (s1, s2)
const char *s1, *s2;
```

```
size_t strspn (s1, s2)
const char *s1, *s2;
```

```
size_t strcspn (s1, s2)
const char *s1, *s2;
```

```
char *strtok (s1, s2)
const char *s1, *s2;
```

## Description

---

The arguments *s1*, *s2*, and *s* point to strings (arrays of characters terminated by a null character). The functions *strcat*, *strncat*, *strcpy*, and *strncpy* all alter *s1*. These functions do not check for overflow of the array pointed to by *s1*.

*strcat* appends a copy of string *s2* to the end of string *s1*.

*strdup* returns a pointer to a new string which is a duplicate of the string pointed to by *s1*. The space for the new string is obtained using *malloc*(*S*). If the new string cannot be created, null is returned.

*strncat* appends at most *n* characters. Each returns a pointer to the null-terminated result.

*strcmp* compares its arguments and returns an integer less than, equal to, or greater than 0, according as *s1* is lexicographically less than, equal to, or greater than *s2*. *strncmp* makes the same comparison but looks at most *n* characters.

*strcpy* copies string *s2* to *s1*, stopping after the null character has been copied. *strncpy* copies exactly *n* characters, truncating *s2* or adding null characters to *s1* if necessary. The result will not be null-terminated if the length of *s2* is *n* or more. Each function returns *s1*.

*strlen* returns the number of characters in *s*, not including the terminating null character.

*strchr* (*strrchr*) returns a pointer to the first (last) occurrence of character *c* in string *s*, or a NULL pointer if *c* does not occur in the string. The null character terminating a string is considered to be part of the string.

*strpbrk* returns a pointer to the first occurrence in string *s1* of any character from string *s2*, or a NULL pointer if no character from *s2* exists in *s1*.



*strspn* (*strcspn*) returns the length of the initial segment of string *s1* which consists entirely of characters from (not from) string *s2*.

*strtok* considers the string *s1* to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string *s2*. The first call (with pointer *s1* specified) returns a pointer to the first character of the first token, and will have written a null character into *s1* immediately following the returned token. The function keeps track of its position in the string between separate calls, so that subsequent calls (which must be made with the first argument a NULL pointer) will work through the string *s1* immediately following that token. In this way subsequent calls will work through the string *s1* until no tokens remain. The separator string *s2* may be different from call to call. When no token remains in *s1*, a NULL pointer is returned.

For user convenience, all these functions are declared in the optional `<string.h>` header file.

## See Also

---

`malloc(S)`

## Notes

---

*strcmp* and *strncmp* are implemented by using the most natural character comparison on the machine. Thus the sign of the value returned when one of the characters has its high-order bit set is not the same in all implementations and should not be relied upon.

Character movement is performed differently in different implementations. Thus overlapping moves may yield surprises.

## Standards Conformance

---

*strcat*, *strchr*, *strcmp*, *strcpy*, *strcspn*, *strlen*, *strncat*, *strncmp*, *strpbrk*, *strspn* and *strtok* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

*strdup* is conformant with:

AT&T SVID Issue 2, Select Code 307-127.

STRING (S)

STRING (S)

*strncpy* and *strrchr* are conformant with:

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;

and NIST FIPS 151-1.

## strtod, atof

convert string to double-precision number

### Syntax

```
double strtod (str, ptr)
char *str, **ptr;
```

```
double atof (str)
char *str;
```

### Description

The *strtod* function returns as a double-precision floating-point number the value represented by the character string pointed to by *str*. The string is scanned up to the first unrecognized character.

The *strtod* function recognizes an optional string of “white-space” characters (as defined by *isspace* in *ctype*(S)), then an optional sign, then a string of digits optionally containing a decimal point, then an optional *e* or *E* followed by an optional sign or space, followed by an integer.

If the value of *ptr* is not (char \*\*)NULL, a pointer to the character terminating the scan is returned in the location pointed to by *ptr*. If no number can be formed, *\*ptr* is set to *str*, and zero is returned.

*atof*(*str*) is equivalent to *strtod*(*str*, (char \*\*)NULL).

### See Also

*ctype*(S), *scanf*(S), *strtol*(S)

### Diagnostics

If the correct value would cause overflow, **±HUGE** (as defined in *<math.h>*) is returned (according to the sign of the value), and *errno* is set to **ERANGE**.

If the correct value would cause underflow, zero is returned and *errno* is set to **ERANGE**.



## Standards Conformance

---

*atof* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

*strtod* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

and ANSI X3.159-198X C Language Draft Standard, May 13, 1988.

## strtol, atol, atoi

---

convert string to integer

### Syntax

---

```
long strtol (str, ptr, base)
char *str, **ptr;
int base;
```

```
long atol (str)
char *str;
```

```
int atoi (str)
char *str;
```

### Description

---

The *strtol* function returns as a long integer the value represented by the character string pointed to by *str*. The string is scanned up to the first character inconsistent with the base. Leading “white-space” characters (as defined by *isspace* in *ctype*(S)) are ignored.

If the value of *ptr* is not (char \*\*)NULL, a pointer to the character terminating the scan is returned in the location pointed to by *ptr*. If no integer can be formed, that location is set to *str*, and zero is returned.

If *base* is positive (and not greater than 36), it is used as the base for conversion. After an optional leading sign, leading zeros are ignored, and “0x” or “0X” is ignored if *base* is 16.

If *base* is zero, the string itself determines the base. After an optional leading sign a leading zero indicates octal conversion, and a leading “0x” or “0X” hexadecimal conversion. Otherwise, decimal conversion is used.

Truncation from long to int can, of course, take place upon assignment or by an explicit cast.

*atol(str)* is equivalent to *strtol(str, (char \*\*)NULL, 10)*.

*atoi(str)* is equivalent to *(int) strtol(str, (char \*\*)NULL, 10)*.

### See Also

---

*ctype*(S), *scanf*(S), *strtod*(S)

## Diagnostics

---

If the argument *ptr* is a null-pointer, the function *strtol* will return the value of the string *str* as a long integer.

If the argument *ptr* is not NULL, the function *strtol* will return the value of the string *str* as a long integer, and a pointer to the character terminating the scan will be returned in the location pointed to by *ptr*.

If no integer can be formed, that location is set to the argument *str* and the function *strtol* returns 0.

## Note

---

Overflow conditions are ignored.

## Standards Conformance

---

*atoi* and *atol* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
ANSI X3.159-198X C Language Draft Standard, May 13, 1988;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

*strtol* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
and ANSI X3.159-198X C Language Draft Standard, May 13, 1988.



## subsystems

---

manipulation routines for Subsystems database

### Syntax

---

```
int authorized_user (auth)
char *auth;
```

```
int total_auths()
```

```
int widest_auth()
```

```
int primary_auth (name)
char *name;
```

```
int secondary_auth (name)
char *name;
```

```
char *primary_of_secondary_auth (name)
char *name;
```

```
int write_authorizations (user, auth_list, list_len)
char *user;
char **auth_list;
int list_len;
```

### Description

---

These routines manipulate and refer to the Subsystems database, which is extracted from the Protected Password entries for all users. The Subsystems database stores primary and secondary authorizations for users with respect to protected subsystems on the Security Module Package. A protected subsystem is associated with a special group, and provides some service to users. It is called protected because all its programs run SGID to that group, and the files and devices that the subsystem references are only accessible to that group, and therefore through the protected subsystem's programs.

A *primary authorization* for a protected subsystem allows a user to assume the administrator role for that subsystem. The primary authorization name is the group name whose effective identity is used by the subsystem programs to protect subsystem files. If a user possesses a primary authorization for a subsystem, he/she can do all actions within that subsystem. The list of primary authorizations is as follows:

## SUBSYSTEMS (S)

## SUBSYSTEMS (S)

|          |                                                                          |
|----------|--------------------------------------------------------------------------|
| audit    | Audit administrator.                                                     |
| auth     | Authentication database administrator.                                   |
| backup   | Backup and file system maintenance administrator.                        |
| cron     | Cron subsystem administrator (at, batch, crontab).                       |
| lp       | Line printer spooling subsystem administrator.                           |
| mem      | Authorization to view information about other users (memory devices).    |
| sysadmin | System administrator functions that require root privilege.              |
| terminal | The authorization to send unfiltered information between user terminals. |
| uucp     | The ability to run uucp (not currently supported).                       |

A *secondary authorization* allows finer grain operations within protected subsystems. These authorizations are often granted either to specific users or to the entire user community by appropriate setup of user Protected Password entries and the Defaults database. Each secondary authorization is associated with exactly one protected subsystem, and only allows operations with respect to that subsystem. The secondary authorizations are as follows:

|             |                                                                                                   |
|-------------|---------------------------------------------------------------------------------------------------|
| printqueue  | Authorization to see other user's jobs in the print queues (lp).                                  |
| printerstat | Authorization to change printer status with <i>enable(C)</i> and <i>disable(C)</i> commands (lp). |
| queryspace  | Authorization to use <i>df(C)</i> to query file system space (backup).                            |

*Authorized\_user* returns non-zero if the login user ID associated with the current process has the specified authorization in the subsystem which is currently running. This subsystem is identified by the effective group under which the program began execution. The primary authorization for any subsystem grants all secondary authorizations for that subsystem.

The rest of the routines are for programs which will manipulate user entries directly, or which have to validate authorizations input by users. They may not be supported in future releases of the SMP. *Total\_auths* returns the number of primary and secondary



authorizations recognized by the system. *Widest\_auth* returns the longest string name of any authorization. *Primary\_auth* maps a primary authorization name to a bit offset in a mask. This is used for checking a user's authorizations against the Protected Password entry. *Secondary\_auth* maps a secondary authorization to a bit offset. *Primary\_of\_secondary\_auth* returns the primary authorization name associated with the secondary authorization. The primary authorization name is also the subsystem name in which the secondary authorization is recognized.

*Write\_authorizations* updates the Subsystems database for a given user. It takes *list\_len* authorizations from the *auth\_list* array of string pointers and associates those authorizations with the given user. If the first character string pointer references the string "default," the user is given default authorizations. This routine is used by the screen-oriented routine which updates the user's Protected Password entry, and then must propagate the user's authorizations to the Subsystems database.

---

## See Also

getprpwent(S)

---

## Diagnostics

*Authorized\_user* return non-zero if the user possesses the specified authorization, otherwise zero. *Total\_auths* returns the sum of the number of primary authorizations and the number of secondary authorizations recognized by the system. *Widest\_auth* returns the length of the longest string name of an authorization. This length does not include the trailing NULL character. Both *primary\_auth* and *secondary\_auth* return a negative value if the authorization name is not supported; otherwise, they return the bit offset of the authorization. The Protected Password database stores both types of authorizations in the same mask. *Primary\_of\_secondary\_auth* returns a pointer to a static area containing the primary authorization name associated with the secondary authorization. The string must be copied if it is to be modified. *Write\_authorizations* returns 0 on success, non-zero on permission failures, I/O errors, etc.

---

## Notes

Programs using this routine must be compiled with *-lprot*.



## swab

---

swap bytes

### Syntax

---

```
void swab (from, to, nbytes)
char *from, *to;
int nbytes;
```

### Description

---

The *swab* function copies *nbytes* bytes pointed to by *from* to the array pointed to by *to*, exchanging adjacent even and odd bytes. *nbytes* should be even and non-negative. If *nbytes* is odd and positive, *swab* uses *nbytes*-1 instead. If *nbytes* is negative, *swab* does nothing.

### Standards Conformance

---

*swab* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## **sync**

---

update super block

### **Syntax**

---

`void sync ( )`

### **Description**

---

The *sync* system call causes all information in memory that should be on disk to be written out. This includes modified super blocks, modified inodes, and delayed block I/O.

It should be used by programs which examine a file system, for example, *fsck*, *df*, etc. It is mandatory before a re-boot.

The writing, although scheduled, is not necessarily complete upon return from *sync*.

### **Standards Conformance**

---

*sync* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## sysconf

---

get configurable system variables

### Syntax

---

```
#include <unistd.h>
```

```
long sysconf (name)
int name ;
```

### Description

---

The *sysconf()* function provides a method for the application to determine the current value of a configurable system limit or option (*variable*).

The name argument represents the system variable to be queried. The implementation shall support all of the variables listed in the following table and may support others. The variables in the following table come from <limits.h> or <unistd.h> (or <time.h>) from CLK\_TCK, and the symbolic constants, defined in <unistd.h>, that are the corresponding values used for *name*.

The value of CLK\_TCK is evaluated at run-time. The value returned by *sysconf()* for \_SC\_CLK\_TCK is the same as that returned by CLK\_TCK.

### Return Value

---

If *name* is an invalid value, *sysconf()* returns a -1. If the variable corresponding to *name* is not defined on the system, *sysconf()* returns -1 without changing the value of *errno*.

Otherwise, the *sysconf()* function returns the current variable value on the system. The value returned is not more restrictive than the corresponding value described to the application when it was compiled with the implementation's <limits.h> or <unistd.h>. The value does not change during the lifetime of the calling process.



| Variable             | name Value        |
|----------------------|-------------------|
| {ARG_MAX}            | {_SC_ARG_MAX}     |
| {CHILD_MAX}          | {_SC_CHILD_MAX}   |
| {CLK_TCK}            | {_SC_CLK_TCK}     |
| {NGROUPS_MAX}        | {_SC_NGROUPS_MAX} |
| {OPEN_MAX}           | {_SC_OPEN_MAX}    |
| {_POSIX_JOB_CONTROL} | {_SC_JOB_CONTROL} |
| {_POSIX_SAVED_IDS}   | {_SC_SAVED_IDS}   |
| {_POSIX_VERSION}     | {_SC_VERSION}     |

## Diagnostics

---

If any of the following conditions occur, the *sysconf()* function returns -1 and sets *errno* to the corresponding value:

[EINVAL] The value of the name argument is invalid.

## Standards Conformance

---

*sysconf* is conformant with:

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

## sysfs

---

get file system type information

### Syntax

---

```
#include <sys/fstyp.h>
#include <sys/fsid.h>
```

```
int sysfs (opcode, fsname)
int opcode;
char *fsname;
```

```
int sysfs (opcode, fs_index, buf)
int opcode;
int fs_index;
char *buf;
```

```
int sysfs (opcode)
int opcode;
```

### Description

---

The *sysfs* system call returns information about the file system types configured in the system. The number of arguments accepted by *sysfs* varies and depends on the *opcode*. The currently recognized *opcodes* and their functions are described below:

|                   |                                                                                                                                                                                                                                                              |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>GETFSIND</b>   | translates <i>fsname</i> , a null-terminated file-system identifier, into a file-system type index.                                                                                                                                                          |
| <b>GETFSTYP</b>   | translates <i>fs_index</i> , a file-system type index, into a null-terminated file-system identifier and writes it into the buffer pointed to by <i>buf</i> ; this buffer must be at least of size <b>FSTYPSZ</b> as defined in <i>&lt;sys/fstyp.h&gt;</i> . |
| <b>GETNFSSTYP</b> | returns the total number of file system types configured in the system.                                                                                                                                                                                      |

The *sysfs* system call will fail if one or more of the following is true:

|          |                                                                                                                           |
|----------|---------------------------------------------------------------------------------------------------------------------------|
| [EINVAL] | <i>fsname</i> points to an invalid file-system identifier; <i>fs_index</i> is zero, or invalid; <i>opcode</i> is invalid. |
| [EFAULT] | <i>buf</i> or <i>fsname</i> point to an invalid user address.                                                             |

## Diagnostics

---

Upon successful completion, *sysfs* returns the file-system type index if the *opcode* is **GETFSIND**, a value of 0 if the *opcode* is **GETFSTYP**, or the number of file system types configured if the *opcode* is **GETNFS-TYP**. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.



# sysi86

## machine-specific functions

### Syntax

```
#include <sys/sysi86.h>
```

```
long sysi86(cmd,arg)
int cmd ;
```

### Description

The *sysi86* system call implements machine-specific functions. The *cmd* argument determines the function to be performed. The types of the arguments expected depend on the function.

#### Command RTODC

When *cmd* is RTODC, the expected argument is the address of a *struct rtc\_t* (from the header file <sys/rtc.h>):

```
struct rtc_t {
 char rtc_sec, rtc_asec, rtc_min, rtc_amin,
 rtc_hr, rtc_ahr, rtc_dow, rtc_dom,
 rtc_mon, rtc_yr, rtc_statusg,
 rtc_statusb, rtc_statusc, rtc_statusd;
};
```

This function reads the hardware time-of-day clock and returns the data in the structure referenced by the argument. This command is available only to the super-user.

#### RDUBLK

This command reads the u-block (per process user information as defined by *structuser* in the <sys/user> header file) for a given process. When *cmd* is RDUBLK, *sysi86* takes three additional arguments: the process ID, the address of a buffer, and the number of bytes to read:

```
sysi86(RDUBLK, pid, buf, n)
 int pid;
 char *buf;
 ind n;
```

**Command SI86FPHW**

This command expects the address of an integer as its argument. After successful return from the system call, the integer specifies how floating-point computation is supported.

The low-order byte of the integer contains the value of "fpkind", a variable that specifies whether an 80287 or 80387 floating-point coprocessor is present, emulated in software, or not supported. The values are defined in the header file <sys/fp.h>.

|        |                                         |
|--------|-----------------------------------------|
| FP_NO  | no fp chip, no emulator (no fp support) |
| FP_SW  | no fp chip, using software emulator     |
| FP_HW  | chip present bit                        |
| FP_287 | 80287 chip present                      |
| FP_387 | 80387 chip present                      |

**Command SETName**

This command, which is only available to the super-user, expects an argument of type *char \** which points to a NULL terminated string of at most 7 characters. The command will change the running system's *sysname* and *nodename* (see *uname(S)*) to this string.

**Command STIME**

When *cmd* is STIME, an argument of type long is expected. This function sets the system time and date (not the hardware clock). The argument contains the time as measured in seconds from 00:00:00 GMT January 1, 1970. Note that this command is only available to the super-user.

**Command SI86DSR**

This command sets a segment or gate descriptor in the kernel. The following descriptor types are accepted:

- executable and data segments in the LDT at DPL 3
- a call gate in the GDT at DPL 3 that points to a segment in the LDT

The argument is a pointer to a request structure that contains the values to be placed in the descriptor. The request structure is declared in the <sys/sysi86.h> header file.

**Command SI86MEM**

This command returns the size of available memory in bytes.

## Command SI86SWPI

When *cmd* is SI86SWPI, individual swapping areas may be added, deleted or the current areas determined. The address of an appropriately primed swap buffer is passed as the only argument. (Refer to *sys/swap.h* header file for details of loading the buffer.)

The format of the swap buffer is:

```
struct swapint {
 char si_cmd; /* command: SI_LIST, SI_ADD, SI_DEL */
 char *si_buf; /* swap file path pointer */
 int si_swpl0; /* start block */
 int si_nblks; /* swap size */
};
```

Note that the add and delete options of the command may only be exercised by the super-user.

Typically, a swap area is added by a single call to *sysi86*. First, the swap buffer is primed with appropriate entries for the structure members. Then *sysi86* is invoked.

```
#include <sys/sysi86.h>
#include <sys/swap.h>

struct swapint swapbuf; /*swap into buffer ptr*/

sysi86(SI86SWPI, &swapbuf);
```

If this command succeeds, it returns 0 to the calling process. This command fails, returning -1, if one or more of the following is true:

- [EFAULT]        *swapbuf* points to an invalid address
- [EFAULT]        *swapbuf.si\_buf* points to an invalid address
- [ENOTBLK]       Swap area specified is not a block special device
- [EEXIST]        Swap area specified has already been added
- [ENOSPC]        Too many swap areas in use (if adding)
- [ENOMEM]        Tried to delete last remaining swap area
- [EINVAL]        Bad arguments
- [ENOMEM]        No place to put swapped pages when deleting a swap area



## Return Value

---

Upon error, -1 is returned and *errno* is set to indicate the error. When the *cmd* is invalid, *errno* is set to EINVAL.

## See Also

---

uname(S), swap(ADM)

## system

---

issue a shell command

### Syntax

---

```
#include <stdio.h>
```

```
int system (string)
const char *string;
```

### Description

---

The *system* function causes the *string* to be given to *sh*(C) as input, as if the string had been typed as a command at a terminal. The current process waits until the shell has completed, then returns the exit status of the shell.

### Files

---

/bin/sh

### See Also

---

exec(S), sh(C)

### Diagnostics

---

The *system* function forks to create a child process that in turn exec's /bin/sh in order to execute *string*. If the fork or exec fails, *system* returns a negative value and sets *errno*.

### Standards Conformance

---

*system* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
and ANSI X3.159-198X C Language Draft Standard, May 13, 1988.

## tam

---

### TAM transition libraries

## Syntax

---

```
#include <tam.h>
```

```
cc -I /usr/include/tam [flags] files -ltam -lcurses [libraries]
```

## Description

---

These routines are used to convert existing TAM programs such that they run on the 3B processor line using any terminal supported by **curses**, the low-level ETI library. Once you change a TAM program to remove machine-specific code, you then recompile it **#include**ing the standard TAM header file **tam.h** and link it with the TAM transition and **curses** libraries.

## Functions

---

The following is a list of TAM routines supplied in the transition library. Those routines marked with a dagger (†) are macros and don't return any value.

```
addch(c)†
```

```
char c;
```

See *curses*(S)

```
addstr(s)†
```

```
char *s;
```

```
int adf_gttok(ptr, tbl)
```

```
char *ptr;
```

```
struct s_kwtbl *tbl;
```

See *paste*(CT)

```
char *adf_gtwrd(sptr, dptr)
```

```
char *sptr, *dptr;
```

```
char *adf_gtxcd(sptr, dptr)
```

```
char *sptr, *dptr;
```

```
int attroff(attrs)
```

```
long attrs;
```

See *curses*(S)

```
int attron(attrs)
```

```
long attrs;
```



**int baudrate()**

**int beep()**

**int cbreak()**

**int clear()**

**clearok(dummy, dummy)†**  
**int dummy;**

**int clrtobot()**

**int clrtoeol()**

**int delch()**

**int deleteln()**

**int echo()**

**int endwin()**

**erase()†**

**int exhelp (hfile, htitle)**  
**char \*hfile, \*htitle;**

**int fixterm()**

*See curses(S)*

**flash()†**

**int flushing()**

**int form (form, op)**  
**form\_t \*form;**  
**int op;**

*See form(S)*

**int getch()**

*See curses(S)*

**getyx(win, r, c)†**  
**int win, r, c;**

**int initscr()**

**int insch(ch)**  
**char ch;**

**int insertln()**

**int iswind()**

See *tam*(S); always returns 0

**char \*kcodemap (code)**

**unsigned char code;**

See *curses*(S)

**int keypad (dummy, flag)**

**int dummy, flag;**

**leaveok(dummy, dummy)†**

**int dummy;**

**int menu (menu, op)**

**menu\_t \*menu;**

**int op;**

See *menu*(S)

**int message (mtype, hfile, htitle, format [, arg ...])**

**int mtype;**

**char \*hfile, \*htitle, \*format;**

**move(r, c)†**

**int r, c;**

See *curses*(S)

**mvaddch(r, c, ch)†**

**int r, c;**

**char ch;**

**mvaddstr(r, c, s)†**

**int r, c;**

**char \*s;**

**unsigned long mvinch(r, c)**

**int r, c;**

**nl()†**

**int nocbreak()**

**int nodelay(dummy, bool)**

**int dummy, bool;**

**int noecho()**

**nonl()†**

**int pb\_check (stream)**

**FILE \*stream;**

See *paste*(CT)

```
int pb_empty (stream)
FILE *stream;
```

```
int pb_gbuf (ptr, n, fn, stream)
char *ptr;
int n;
int (*fn) ();
FILE *stream;
```

```
char *pb_gets (ptr, n, stream)
char *ptr;
int n;
FILE *stream;
```

```
char *pb_name()
FILE *pb_open()
```

```
int pb_puts (ptr, stream)
char *ptr;
FILE *stream;
```

```
int pb_seek (stream)
FILE *stream;
```

```
int pb_wEOF (stream)
FILE *stream;
```

```
int printw (fmt[, arg1 ... argn])
char *fmt;
```

See *curses*(S)

```
refresh()†
```

```
int resetterm()
```

```
int resetty()
```

```
int savetty()
```

```
int track (w, trk, op, butptr, whyptr)
int w, op, *butptr, *whyptr;
track_t *trk;
```

See *wgetc*()

```
int wcmd (wn, cp)
short wn;
char *cp;
```

See *tam*(S). Outputs a null-terminated string to the entry/echo line.

```
int wcreate (row, col, height, width, flags)
```



**short row, col, height, width;**  
**unsigned short flags;**  
 Creates a window.

**int wdelete (wn)**  
**short wn;**  
 Deletes the specified window.

**void wexit(ret)**  
**int ret;**  
 See *tam(S)*

**int wgetc (wn)**  
**short wn;**

**int wgetmouse (wn, ms)**  
**short wn;**  
**struct umdata \*ms;**  
 no-op; returns 0.

**int wgetpos (wn, rowp, colp)**  
**short wn;**  
**int \*rowp, \*colp;**  
 Gets the current position (row, column)  
 of the cursor in the specified window (*wn*).

**int wgetsel()**  
 Returns the currently selected window.

**int wgetstat (wn, wstatp)**  
**short wn;**  
**WSTAT \*wstatp;**  
 Returns the information in WSTAT for a window.

**int wgoto (wn, row, col)**  
**short wn, row, col;**  
 Moves the window's cursor to a specified row, column.

**void wicoff (wn, row, col, icp)**  
**short wn, row, col;**  
**struct icon \*icp;**  
 no-op. returns 0.

**void wicon (wn, row, col, icp)**  
**short wn, row, col;**  
**struct icon \*icp;**  
 no-op. returns 0.

**int wind (type, height, width, flags, pfont)**  
**int type, height, width;**  
**short flags;**  
**char \*pfont[];**

**void winit()**

Sets up the process for window access. See *tam(S)*.

**int wlabel (wn, cp)**

**short wn;**

**char \*cp;**

Outputs a null-terminated string to the window label area.

**int wndelay (wn, bool)**

**int wn, bool;**

**void wnl (wn, flag)**

**short wn;**

**int flag;**

**int wpostwait()**

Reverses the effects of *wprexec()*.

**int wprexec()**

Performs the appropriate actions for passing a window to a child process.

**int wprintf (wn, fmt[, arg1 ... argn])**

**short wn;**

**char \*fmt;**

**int wprompt (wn, cp)**

**short wn;**

**char \*cp;**

Outputs a null-terminated string to the prompt line.

**int wputc (wn, c)**

**short wn;**

**char c;**

Outputs a character to a window (*wn*).

**int wputs (wn, cp)**

**short wn;**

**char \*cp;**

Outputs a character string to a window.

**int wrastop (w, srcbase, srcwidth, dstbase,**

**dstwidth, srcx, srcy, dstx,**

**dsty, width, height, srcop,**

**dstop, pattern)**

**int w;**

**unsigned short \*srcbase, \*dstbase, \*pattern;**

**unsigned short srcwidth, dstwidth, width, height;**

**unsigned short srcx, srcy, dstx, dsty;**

**char srcop, dstop;**

**int wreadmouse (wn, xp, yp, bp, rp)**  
**short wn;**  
**int \*xp, \*yp, \*bp, \*rp;**  
 no-op; returns 0.

**int wrefresh (wn)**  
**short wn;**  
 Flushes all output to the window.

**int wselect (wn)**  
**short wn;**  
 Selects the specified window as the current or active one.

**int wsetmouse (wn, ms)**  
**short wn;**  
**struct umdata \*ms;**  
 no-op; returns 0.

**int wsetstat (wn, wstatp)**  
**short wn;**  
**WSTAT \*wstatp;**  
 Sets the status for a window.

**int wslk (wn, 0, slong1, slong2, sshort)**  
**short wn;**  
**char \*slong1, \*slong2, \*sshort;**  
 Writes a null-terminated to a set of screen labeled keys.

**int wslk (wn, kn, llabel, slabel)**  
**short wn, kn;**  
**char \*llabel, \*slabel;**  
 Writes a null-terminated string to a screen labeled key.  
 The alternate form writes all the screen labeled keys at once,  
 which is more efficient.

**int wuser (wn, cp)**  
**short wn;**  
**char \*cp;**

## See Also

---

curses(S), field(S), fieldtype(S), form(S), item(S), menu(S), panel(S)



# tcgetattr, tcsetattr

---

## state functions

### Syntax

---

```
#include <termios.h>
```

```
int tcgetattr (fildes, termios_p);
int fildes;
struct termios *termios_p;
```

```
int tcsetattr (fildes, optional_actions, termios_p);
int fildes, optional_actions;
struct termios *termios_p;
```

### Description

---

The *tcgetattr()* function gets the parameters associated with the object referred to by *fildes* and store them in the *termios* structure referenced by *termios\_p*. This function is allowed from a background process; however the terminal attributes may subsequently be changed by a foreground process.

The *tcsetattr()* function shall set the parameters associated with the terminal (unless support is required from the underlying hardware that is not available) from the *termios* structure referenced by *termios\_p* as follows:

1. If *optional\_actions* is TCSANOW, the change occurs immediately.
2. If *optional\_actions* is TCSADRAIN, the change occurs after all output written to *fildes* has been transmitted. This function should be used when changing parameters that affect output.
3. If *optional\_actions* is TCSAFLUSH, the change occurs after all output written to *fildes* has been transmitted and all input received but not read is discarded.

The symbolic constants for the values of the *optional\_actions* are defined in *<termios.h>*.

### Return Value

---

Upon successful completion, these routines return a value of 0. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## Errors

---

If any of the following conditions occur, *tcgetattr()* returns -1 and sets *errno* to the corresponding value:

- EBADF      The *fildev* argument is not a valid file descriptor.  
ENOTTY     The file associated with *fildev* is not a terminal.

If any of the following conditions occur, *tcsetattr()* returns -1 and sets *errno* to the corresponding value:

- EBADF      The *fildev* argument is not a valid file descriptor.  
EINVAL     The *optional\_actions* argument is not a proper value, or an attempt was made to change an attribute represented in the *termios* structure to an unsupported value.  
ENOTTY     The file associated with *fildev* is not a terminal.

## See Also

---

*tcflow*(S)

## Standards Conformance

---

*tcgetattr* and *tcsetattr* are conformant with:  
AT&T SVID Issue 2, Select Code 307-127.

# **tcdrain, tcflow, tcflush, tcsendbreak**

## line control functions

### Syntax

---

```
#include <termios.h>
```

```
int tcdrain (fildes)
int fildes;
```

```
int tcflow (fildes, action)
int fildes;
int action;
```

```
int tcflush (fildes, queue_selector)
int fildes;
int queue_selector;
```

```
int tcsendbreak (fildes, duration)
int fildes;
int duration;
```

### Description

---

If the terminal is using asynchronous serial data transmission, the *tcsendbreak()* function causes transmission of a continuous stream of zero-valued bits for a specific duration. If *duration* is zero, *tcsendbreak* transmits zero-valued bits for at least .25 seconds and not more than .5 seconds. If *duration* is not set to zero, *tcsendbreak* sends zero-valued bits for the amount of time specified by the implementation.

If the terminal is not using asynchronous serial data transmission, it is implementation defined whether the *tcsendbreak* function sends data to generate a break condition (as defined by the implementation) or returns without taking any action.

The *tcdrain()* function waits until all output written to *fildes* has been transmitted.

The *tcflush()* function discards all data written to *fildes* but not transmitted, or data received but not read, depending on the value of *queue\_selector*:

1. If *queue\_selector* is TCIFLUSH, it flushes data received but not read.
2. If *queue\_selector* is TCOFLUSH, it flushes data written but not transmitted.



3. If *queue\_selector* is TCIOFLUSH, it performs both actions.

The *tcflow()* function suspends transmission or reception of data on the object referred to by *fildev*, depending on the value of *action*. Possible values for *action* include:

1. If *action* is TCOOFF, output is suspended.
2. If *action* is TCOON, output is restarted.
3. If *action* is TCIOFF, a STOP character is transmitted, which is intended to cause the terminal to stop transmitting data to the system.
4. If *action* is TCION, a START character is transmitted, which is intended to cause the terminal to start transmitting data to the system.

The symbolic constants for the values of *queue\_selector* and *action* are defined in `<termios.h>`. The default on open of a terminal file is that neither its input nor its output is suspended.

## Return Value

---

Upon successful completion, these routines return a value of zero. Otherwise, a value of -1 is returned and *errno* is set to indicate the error. Possible error conditions include:

### *tcsendbreak:*

- [EBADF] The *fildev* argument is not a valid file descriptor.
- [ENOTTY] The file associated with *fildev* is not a terminal.

### *tcdrain:*

- [EBADF] The *fildev* argument is not a valid file descriptor.
- [EINTR] A signal interrupted the function.
- [ENOTTY] The file associated with *fildev* is not a terminal.

### *tcflush:*

- [EBADF] The *fildev* argument is not a valid file descriptor.
- [EINVAL] The *queue\_selector* argument is not a proper value.
- [ENOTTY] The file associated with *fildev* is not a terminal.

### *tcflow:*

- [EBADF] The *fildev* argument is not a valid file descriptor.
- [EINVAL] The *action* argument is not a proper value.
- [ENOTTY] The file associated with *fildev* is not a terminal.

## See Also

---

*tcattr*(S)

## **Standards Conformance**

---

*tcdrain*, *tcflow*, *tcflush* and *tcsendbreak* are conformant with:  
AT&T SVID Issue 2, Select Code 307-127.

## tcgetpgrp, tcsetpgrp

---

### process group id functions

#### Syntax

---

```
#include <sys/types.h>

pid_t tcgetpgrp (files);
int files;
#include <sys/types.h>

int tcsetpgrp (files, pgrp_id);
int files;
pid_t pgrp_id;
```

#### Description

---

These routines identify and set the parent process group id when job control is defined. The **tcgetpgrp()** function returns the value of the process group id of the foreground process group associated with the terminal.

**tcgetpgrp()** is allowed only from a process that is part of a background process group. However, this information can be changed by a process that is part of the foreground process group by means of the **tcsetpgrp()** call.

**tcsetpgrp()** sets the foreground process id group associated with the terminal to the value of *pgrp\_id*. *files* must be the file associated with the controlling terminal of the calling process. The controlling terminal must also be currently associated with the session of the calling process. The value of *pgrp\_id* must match a process group id of a process in the same session as the calling process. Any other value of *pgrp\_id* will cause an error.

#### Return Value

---

If successful, **tcgetpgrp()** returns the process id of the foreground process group associated with the calling terminal. Otherwise, -1 is returned and *errno* is set to indicate the error.

**tcsetpgrp()** returns a value of zero upon success. Otherwise, -1 is returned and *errno* is set to indicate the error.



## Errors

---

If any of the following conditions occur, *tcgetpgrp()* returns -1 and sets *errno* to the corresponding value:

- |        |                                                                                                                                  |
|--------|----------------------------------------------------------------------------------------------------------------------------------|
| EBADF  | The <i>fildev</i> argument is not a valid file descriptor.                                                                       |
| ENOSYS | The <b>tcgetpgrp()</b> function is not supported.                                                                                |
| ENOTTY | The calling process does not have a controlling terminal or the file described in <i>fildev</i> is not the controlling terminal. |

If any of the following conditions occur, *tcsetpgrp()* returns -1 and sets *errno* to the corresponding value:

- |        |                                                                                                                                                                         |
|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| EBADF  | The <i>fildev</i> argument is not a valid file descriptor.                                                                                                              |
| EINVAL | The value of <i>pgrp_id</i> is not supported.                                                                                                                           |
| ENOSYS | The <b>tcsetpgrp()</b> function is not supported.                                                                                                                       |
| ENOTTY | The calling process does not have a controlling terminal or the file described in <i>fildev</i> is not the controlling terminal.                                        |
| EPERM  | The value of <i>pgrp_id</i> is a value supported by the implementation but does not match the process group id of a process in the same session as the calling process. |

## See Also

---

*tcflow(S)*, *tcattr(S)*,

## Standards Conformance

---

*tcgetpgrp* and *tcsetpgrp* are conformant with:

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support.

# **tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs**

---

performs terminal functions

## **Syntax**

---

```
char PC;
char *BC;
char *UP;
short ospeed;
```

```
int tgetent(bp, name)
char *bp, *name;
```

```
int tgetnum(id)
char *id;
```

```
int tgetflag(id)
char *id;
```

```
char *
tgetstr(id, area)
char *id, **area;
```

```
char *
tgoto(cm, destcol, destline)
char *cm;
int destcol, destline;
```

```
void tputs(cp, affcnt, outc)
register char *cp;
int affcnt;
int (*outc)();
```

## **Description**

---

These functions extract and use capabilities from the terminal capability data base *termcap* (F). These are low level routines; see *curses* (S) for a higher level package.

*tgetent* extracts the entry for terminal *name* into the buffer at *bp*. *bp* should be a character buffer of size 1024 and must be retained through all subsequent calls to *tgetnum*, *tgetflag*, and *tgetstr*. *tgetent* returns -1 if it cannot open the *termcap* file, 0 if the terminal name given does not have an entry, and 1 if all goes well. It looks in the environment for a TERMCAP variable. If found, and the value does not begin with a slash, and the terminal type *name* is the same as the environment

string *TERM*, the *TERMCAP* string is used instead of reading the *termcap* file. If it does begin with a slash, the string is used as a path-name rather than */etc/termcap*. This can speed up entry into programs that call *tgetent*, as well as to help debug new terminal descriptions or to make one for your terminal if you can't write the file */etc/termcap*.

*tgetnum* gets the numeric value of capability *id*, returning -1 if it is not given for the terminal. *tgetflag* returns 1 if the specified capability is present in the terminal's entry, 0 if it is not. *tgetstr* returns the string value of capability *id*, advancing the *area* pointer. It decodes the abbreviations for this field described in *termcap*(F), except for cursor addressing and padding information.

*tgoto* returns a cursor addressing string decoded from *cm* to go to column *destcol* in line *destline*. It uses the external variables *UP* (from the *up* capability) and *BC* (if *bc* is given rather than *bs*) if necessary to avoid placing *\n*, *Ctrl-D* or *NULL* in the returned string. Programs which call *tgoto* should be sure to turn off the *TAB3* bit (see *tty*(M)), since *tgoto* may now output a tab. Note that programs using *termcap* should turn off *TAB3* anyway since some terminals use *Ctrl-I* for other functions, such as nondestructive space.) If a *%* sequence is given which is not understood, then *tgoto* returns *OOPS*.

*tputs* decodes the leading padding information of the string *cp*; *affcnt* gives the number of lines affected by the operation, or 1 if this is not applicable, *outc* is a routine which is called with each character in turn. The external variable *ospeed* should contain the output speed of the terminal as encoded by *stty*(S). The external variable *PC* should contain a pad character to be used (from the *pc* capability) if a *NULL* is inappropriate.

## Files

---

|                              |                           |
|------------------------------|---------------------------|
| <i>/usr/lib/libtermcap.a</i> | - <i>ltermcap</i> library |
| <i>/etc/termcap</i>          | data base                 |

## See Also

---

*curses*(S), *termcap*(F), *tty*(M)

## Credit

---

This utility was developed at the University of California at Berkeley and is used with permission.

## Notes

---

These routines must be linked by using the *-ltermcap* linker option.



# terminfo

terminal description database.

## Syntax

```
#include <curses.h>
#include <term.h>
```

```
cc -DM_TERMINFO [-DMINICURSES] ... -linfo [-lx]
```

## Description

These routines give the user a method of updating screens with reasonable optimization. In order to initialize the routines, the routine *initscr* must be called before any of the other routines that deal with windows and screens are used. The routine *endwin* should be called before exiting. To get character-at-a-time input without echoing, (most interactive, screen oriented-programs want this) after calling *initscr* you should call “*nonl( ); cbreak( ); noecho( );*”

The full curses interface permits manipulation of data structures called *windows* which can be thought of as two dimensional arrays of characters representing all or part of a CRT screen. A default window called *stdscr* is supplied, and others can be created with *newwin*. Windows are referred to by variables declared “WINDOW \*”, the type WINDOW is defined in *curses.h* to be a C structure. These data structures are manipulated with functions described below, among which the most basic are *move*, and *addch*. (More general versions of these functions are included with names beginning with ‘w’, allowing you to specify a window. The routines not beginning with ‘w’ affect *stdscr*.) Then *refresh()* is called, telling the routines to make the users CRT screen look like *stdscr*.

Mini-Curses is a subset of curses which does not allow manipulation of more than one window. To invoke this subset, use

```
-DMINICURSES
```

as a cc option. Mini-Curses is smaller and faster than full curses.

If the environment variable TERMINFO is defined, any program using curses will check for a local terminal definition before checking in the standard place. For example, if the standard place is */usr/lib/terminfo*, and TERM is set to “vt100”, then normally the compiled file is found in */usr/lib/terminfo/v/vt100*. (The “v” is copied from the first letter of “vt100” to avoid creation of huge directories.) However, if TERMINFO is set to */usr/mark/myterms*, curses will first check */usr/mark/myterms/v/vt100*, and if that fails, will

then check `/usr/lib/terminfo/v/vt100`. This is useful for developing experimental definitions or when write permission in `/usr/lib/terminfo` is not available.

## See Also

---

`terminfo(F)`

## Functions

---

Routines listed here may be called when using the full curses. Those marked with an asterisk may be called when using Mini-Curses.

|                                  |                                                                                                                                         |
|----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>addch(ch)*</code>          | add a character to <i>stdscr</i> (like <i>putchar</i> )<br>(wraps to next line at end of line)                                          |
| <code>addstr(str)*</code>        | calls <code>addch</code> with each character in <i>str</i>                                                                              |
| <code>attroff(attrs)*</code>     | turn off attributes named                                                                                                               |
| <code>attron(attrs)*</code>      | turn on attributes named                                                                                                                |
| <code>attrset(attrs)*</code>     | set current attributes to <i>attrs</i>                                                                                                  |
| <code>baudrate()**</code>        | current terminal speed                                                                                                                  |
| <code>beep()**</code>            | sound beep on terminal                                                                                                                  |
| <code>box(win, vert, hor)</code> | draw a box around edges of <i>win</i><br><i>vert</i> and <i>hor</i> are chars to use for <i>vert</i> .<br>and <i>hor</i> . edges of box |
| <code>clear()</code>             | clear <i>stdscr</i>                                                                                                                     |
| <code>clearok(win, bf)</code>    | clear screen before next redraw of <i>win</i>                                                                                           |
| <code>clrtoebot()</code>         | clear to bottom of <i>stdscr</i>                                                                                                        |
| <code>clrtoeol()</code>          | clear to end of line on <i>stdscr</i>                                                                                                   |
| <code>cbreak()**</code>          | set <code>cbreak</code> mode                                                                                                            |
| <code>delay_output(ms)*</code>   | insert <i>ms</i> millisecond pause in output                                                                                            |
| <code>delch()</code>             | delete a character                                                                                                                      |
| <code>deleteln()</code>          | delete a line                                                                                                                           |
| <code>delwin(win)</code>         | delete <i>win</i>                                                                                                                       |
| <code>doupdate()</code>          | update screen from all <i>wnoutrefresh</i>                                                                                              |
| <code>echo()**</code>            | set echo mode                                                                                                                           |
| <code>endwin()**</code>          | end window modes                                                                                                                        |
| <code>erase()</code>             | erase <i>stdscr</i>                                                                                                                     |
| <code>erasechar()</code>         | return user's erase character                                                                                                           |
| <code>fixterm()</code>           | restore tty to "in curses" state                                                                                                        |
| <code>flash()</code>             | flash screen or beep                                                                                                                    |
| <code>flushinp()**</code>        | throw away any typeahead                                                                                                                |
| <code>getch()**</code>           | get a char from tty                                                                                                                     |
| <code>getstr(str)</code>         | get a string through <i>stdscr</i>                                                                                                      |
| <code>gettmode()</code>          | establish current tty modes                                                                                                             |
| <code>getyx(win, y, x)</code>    | get ( <i>y</i> , <i>x</i> ) co-ordinates                                                                                                |
| <code>has_ic()</code>            | true if terminal can do insert character                                                                                                |
| <code>has_il()</code>            | true if terminal can do insert line                                                                                                     |
| <code>idlok(win, bf)*</code>     | use terminal's insert/delete line if <i>bf</i> != 0                                                                                     |
| <code>inch()</code>              | get char at current ( <i>y</i> , <i>x</i> ) co-ordinates                                                                                |



## TERMINFO (S)

initscr( )\*  
 insch(c)  
 insertln()  
 intrflush(win, bf)  
 keypad(win, bf)  
 killchar()  
 leaveok(win, flag)  
  
 longname()  
 meta(win, flag)\*  
 move(y, x)\*  
 mvaddch(y, x, ch)  
 mvaddstr(y, x, str)  
 mvcur(oldrow, oldcol,  
     newrow, newcol)  
 mvdelch(y, x)  
 mvgetch(y, x)  
 mvgetstr(y, x)  
 mvinch(y, x)  
 mvinsch(y, x, c)  
 mvprintw(y, x, fmt, args)  
 mvscanw(y, x, fmt, args)  
 mvwaddch(win, y, x, ch)  
 mvwaddstr(win, y, x, str)  
 mvwdelch(win, y, x)  
 mvwgetch(win, y, x)  
 mvwgetstr(win, y, x, str)  
 mvwin(win, by, bx)  
 mvwinch(win, y, x)  
 mvwinsch(win, y, x, c)  
 mvwprintw(win, y, x,  
     fmt, args)  
 mvwscanw(win, y, x,  
     fmt, args)  
 newpad(nlines, ncols)  
 newterm(type, fd)  
  
 newwin(lines, cols,  
     begin\_y, begin\_x)  
 nl( )\*  
 nocbreak( )\*  
 nodelay(win, bf)  
 noecho( )\*  
 nonl( )\*  
 noraw( )\*  
 overlay(win1, win2)  
 overwrite(win1, win2)

## TERMINFO (S)

initialize screens  
 insert a char  
 insert a line  
 interrupts flush output if bf is TRUE  
 enable keypad input  
 return current user's kill character  
 OK to leave cursor anywhere after  
 refresh if flag!=0 for win, otherwise cursor  
 must be left at current position.  
 return verbose name of terminal  
 allow meta characters on input if flag != 0  
 move to (y, x) on *stdscr*  
 move(y, x) then addch(ch)  
 similar...  
 low level cursor motion  
  
 like delch, but move(y, x) first  
 etc.  
  
  
 create a new pad with given dimensions  
 set up new terminal of given type to  
 output on fd  
 create a new window  
  
 set newline mapping  
 unset cbreak mode  
 enable nodelay input mode through getch  
 unset echo mode  
 unset newline mapping  
 unset raw mode  
 overlay win1 on win2  
 overwrite win1 on top of win2



pnoutrefresh(pad,  
pminrow, pmincol,  
sminrow, smincol,  
smaxrow, smaxcol)

printw(fmt, arg1, arg2, ...)

raw()\*

refresh()\*

resetterm()\*

resetty()\*

saveterm()\*

savetty()\*

scanw(fmt, arg1, arg2, ...)

scroll(win)

scrollok(win, flag)

set\_term(new)

setscreg(t, b)

setterm(type)

setupterm(term, filenum,  
errret)

standend()\*

standout()\*

subwin(win, lines, cols,  
begin\_y, begin\_x)

touchwin(win)

traceoff()

traceon()

typeahead(fd)

unctrl(ch)\*

waddch(win, ch)

waddstr(win, str)

wattroff(win, attrs)

wattron(win, attrs)

wattrset(win, attrs)

wclear(win)

wclrtoebot(win)

wclrtoeol(win)

wdelch(win, c)

wdeleteln(win)

werase(win)

wgetch(win)

wgetstr(win, str)

winch(win)

winsch(win, c)

winsertln(win)

wmove(win, y, x)

wnoutrefresh(win)

wprintw(win, fmt, arg1,  
arg2, ...)

wrefresh(win)

like prefresh but with no output until  
doupdate called prefresh(pad, pminrow,  
pmincol, sminrow, smincol, smaxrow,  
smaxcol) refresh from pad starting with  
given upper left corner of pad with  
output to given portion of screen

printf on *stdscr*

set raw mode

make current screen look like *stdscr*

set tty modes to "out of curses" state

reset tty flags to stored value

save current modes as "in curses" state

store current tty flags

scanf through *stdscr*

scroll *win* one line

allow terminal to scroll if flag !=0

now talk to terminal new

set user scrolling region to lines *t* through *b*

establish terminal with given type

clear standout mode attribute

set standout mode attribute

create a subwindow

change all of *win*

turn off debugging trace output

turn on debugging trace output

use file descriptor *fd* to check typeahead

printable version of *ch*

add char to *win*

add string to *win*

turn off *attrs* in *win*

turn on *attrs* in *win*

set *attrs* in *win* to *attrs*

clear *win*

clear to bottom of *win*

clear to end of line on *win*

delete char from *win*

delete line from *win*

erase *win*

get a char through *win*

get a string through *win*

get char at current (*y*, *x*) in *win*

insert char into *win*

insert line into *win*

set current (*y*, *x*) co-ordinates on *win*

refresh but no screen output

printf on *win*

make screen look like *win*

|                                      |                                        |
|--------------------------------------|----------------------------------------|
| wscanw(win, fmt, arg1,<br>arg2, ...) | scanf through <i>win</i>               |
| wsetscreg(win, t, b)                 | set scrolling region of <i>win</i>     |
| wstandend(win)                       | clear standout attribute in <i>win</i> |
| wstandout(win)                       | set standout attribute in <i>win</i>   |

## Terminfo Level Routines

---

These routines should be called by programs wishing to deal directly with the terminfo database. Due to the low level of this interface, it is discouraged. Initially, *setupterm* should be called. This will define the set of terminal dependent variables defined in *terminfo(F)*. The include files **curses.h** and **term.h** should be included to get the definitions for these strings, numbers, and flags. Parameterized strings should be passed through *tparm* to instantiate them. All terminfo strings (including the output of *tparm*) should be printed with *tputs* or *putp*. Before exiting, *resetterm* should be called to restore the tty modes. (Programs desiring shell escapes can call *resetterm* before the shell is called and *fixterm* after returning from the shell.)

|                                                         |                                                                                                                                                                                                                                                                                                                                            |
|---------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| fixterm()                                               | restore tty modes for terminfo use (called by <i>setupterm</i> )                                                                                                                                                                                                                                                                           |
| resetterm()<br>setupterm(term, fd, rc)                  | reset tty modes to state before program entry read in database. Terminal type is the character string <i>term</i> , all output is to the system file descriptor <i>fd</i> . A status value is returned in the integer pointed to by <i>rc</i> : 1 is normal. The simplest call would be <i>setupterm(0, 1, 0)</i> which uses all defaults. |
| tparm(str, p1, p2, ..., p9)<br>tputs(str, affcnt, putc) | instantiate string <i>str</i> with parms <i>p<sub>1</sub></i> . apply padding info to string <i>str</i> . <i>affcnt</i> is the number of lines affected, or 1 if not applicable. <i>Putc</i> is a putchar-like function to which the characters are passed, one at a time. Note that the user must supply their own <i>putc</i> function.  |
| putp(str)                                               | handy function that calls <i>tputs</i> (str, 1, putchar)                                                                                                                                                                                                                                                                                   |
| vidputs(attrs, putc)                                    | output the string to put terminal in video attribute mode <i>attrs</i> , which is any combination of the attributes listed below. Chars are passed to putchar-like function <i>putc</i> . Standard <i>putchar</i> can be used here or the user can supply their own <i>putc</i> function.                                                  |
| vidattr(attrs)                                          | Like <i>vidputs</i> but outputs through <i>putchar</i>                                                                                                                                                                                                                                                                                     |



## Termcap Compatibility Routines

---

These routines were included as a conversion aid for programs that use *termcap*(S). Their parameters are the same as used in *termcap*. They are emulated using the *terminfo*(F) database. They may be removed at a later date.

|                                     |                                            |
|-------------------------------------|--------------------------------------------|
| <code>tgetent(bp, name)</code>      | look up termcap entry for name             |
| <code>tgetflag(id)</code>           | get boolean entry for id                   |
| <code>tgetnum(id)</code>            | get numeric entry for id                   |
| <code>tgetstr(id, area)</code>      | get string entry for id                    |
| <code>tgoto(cap, col, row)</code>   | apply parms to given cap                   |
| <code>tputs(cap, affcnt, fn)</code> | apply padding to cap calling fn as putchar |

## Attributes

---

The following video attributes can be passed to the functions *attron*, *attroff*, *attrset*.

|                           |                                   |
|---------------------------|-----------------------------------|
| <code>A_STANDOUT</code>   | Terminal's best highlighting mode |
| <code>A_UNDERLINE</code>  | Underlining                       |
| <code>A_REVERSE</code>    | Reverse video                     |
| <code>A_BLINK</code>      | Blinking                          |
| <code>A_DIM</code>        | Half bright                       |
| <code>A_BOLD</code>       | Extra bright or bold              |
| <code>A_BLANK</code>      | Blanking (invisible)              |
| <code>A_PROTECT</code>    | Protected                         |
| <code>A_ALTCHARSET</code> | Alternate character set           |

## Function Keys

---

The following function keys might be returned by *getch* if *keypad* has been enabled. Note that not all of these are currently supported, due to lack of definitions in *terminfo* or the terminal not transmitting a unique code when the key is pressed.

| <i>Name</i>                | <i>Value</i>              | <i>Key name</i>                          |
|----------------------------|---------------------------|------------------------------------------|
| <code>KEY_BREAK</code>     | 0401                      | break key (unreliable)                   |
| <code>KEY_DOWN</code>      | 0402                      | The four arrow keys ...                  |
| <code>KEY_UP</code>        | 0403                      |                                          |
| <code>KEY_LEFT</code>      | 0404                      |                                          |
| <code>KEY_RIGHT</code>     | 0405                      | ...                                      |
| <code>KEY_HOME</code>      | 0406                      | Home key (upward+left arrow)             |
| <code>KEY_BACKSPACE</code> | 0407                      | backspace (unreliable)                   |
| <code>KEY_F0</code>        | 0410                      | Function keys. Space for 64 is reserved. |
| <code>KEY_F(n)</code>      | <code>(KEY_F0+(n))</code> | Formula for fn.                          |



## TERMINFO (S)

|            |      |
|------------|------|
| KEY_DL     | 0510 |
| KEY_IL     | 0511 |
| KEY_DC     | 0512 |
| KEY_IC     | 0513 |
| KEY_EIC    | 0514 |
| KEY_CLEAR  | 0515 |
| KEY_EOS    | 0516 |
| KEY_EOL    | 0517 |
| KEY_SF     | 0520 |
| KEY_SR     | 0521 |
| KEY_NPAGE  | 0522 |
| KEY_PPAGE  | 0523 |
| KEY_STAB   | 0524 |
| KEY_CTAB   | 0525 |
| KEY_CATAB  | 0526 |
| KEY_ENTER  | 0527 |
| KEY_SRESET | 0530 |
| KEY_RESET  | 0531 |
| KEY_PRINT  | 0532 |
| KEY_LL     | 0533 |

## TERMINFO (S)

|                                   |
|-----------------------------------|
| Delete line                       |
| Insert line                       |
| Delete character                  |
| Insert char or enter insert mode  |
| Exit insert char mode             |
| Clear screen                      |
| Clear to end of screen            |
| Clear to end of line              |
| Scroll 1 line forward             |
| Scroll 1 line backwards (reverse) |
| Next page                         |
| Previous page                     |
| Set tab                           |
| Clear tab                         |
| Clear all tabs                    |
| Enter or send (unreliable)        |
| soft (partial) reset (unreliable) |
| reset or hard reset (unreliable)  |
| print or copy                     |
| home down or bottom (lower left)  |

## time

---

get time

### Syntax

---

```
#include <sys/types.h>
```

```
time_t time (tloc)
long *tloc;
```

### Description

---

The *time* system call returns the value of time in seconds since 00:00:00 Greenwich Mean Time (GMT), January 1, 1970.

If *tloc* is non-zero, the return value is also stored in the location to which *tloc* points.

### See Also

---

stime(S)

### Diagnostics

---

Upon successful completion, *time* returns the value of time. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

### Warning

---

The *time* system call fails and its actions are undefined if *tloc* points to an illegal address.

### Standards Conformance

---

*time* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
ANSI X3.159-198X C Language Draft Standard, May 13, 1988;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

## times

---

get process and child process times

### Syntax

---

```
#include <sys/types.h>
#include <sys/times.h>
```

```
long times (buffer)
struct tms *buffer;
```

### Description

---

The *times* system call fills the structure pointed to by *buffer* with time-accounting information. The following are the contents of this structure:

```
struct tms {
 time_t tms_utime;
 time_t tms_stime;
 time_t tms_cutime;
 time_t tms_cstime;
};
```

This information comes from the calling process and each of its terminated child processes for which it has executed a *wait*. All times are reported in clock ticks per second. Clock ticks are a system-dependent parameter. The specific value for an implementation is defined by the variable *HZ*, found in the include file **param.h**.

*tms\_utime* is the CPU time used while executing instructions in the user space of the calling process.

*tms\_stime* is the CPU time used by the system on behalf of the calling process.

*tms\_cutime* is the sum of the *tms\_utime*s and *tms\_cutime*s of the child processes.

*tms\_cstime* is the sum of the *tms\_stime*s and *tms\_cstime*s of the child processes.

[EFAULT] The *times* system call will fail if *buffer* points to an illegal address.



## See Also

---

`exec(S)`, `fork(S)`, `time(S)`, `wait(S)`

## Diagnostics

---

Upon successful completion, *times* returns the elapsed real time, in clock ticks per second, from an arbitrary point in the past (for example, system start-up time). This point does not change from one invocation of *times* to another. If *times* fails, a -1 is returned and *errno* is set to indicate the error. Clock ticks occur 100 times per second.

## Standards Conformance

---

*times* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

## **tmpfile**

---

create a temporary file

### **Syntax**

---

```
#include <stdio.h>
```

```
FILE *tmpfile ()
```

### **Description**

---

The *tmpfile* function creates a temporary file using a name generated by *tmpnam*(S), and returns a corresponding FILE pointer. If the file cannot be opened, a NULL pointer is returned. The file will automatically be deleted when the process using it terminates. The file is opened for update ("w+").

### **See Also**

---

*creat*(S), *unlink*(S), *fopen*(S), *mktemp*(S), *stdio*(S), *tmpnam*(S)

### **Standards Conformance**

---

*tmpfile* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.

## tmpnam, tmpnam

---

create a name for a temporary file

### Syntax

---

```
#include <stdio.h>
```

```
char *tmpnam (s)
char *s;
```

```
char *tmpnam (dir, pfx)
char *dir, *pfx;
```

### Description

---

These functions generate file names that can safely be used for a temporary file.

The *tmpnam* function always generates a file name using the path-prefix defined as **P\_tmpdir** in the `<stdio.h>` header file. If *s* is NULL, *tmpnam* leaves its result in an internal static area and returns a pointer to that area. The next call to *tmpnam* will destroy the contents of the area. If *s* is not NULL, it is assumed to be the address of an array of at least **L\_tmpnam** bytes, where **L\_tmpnam** is a constant defined in `<stdio.h>`; *tmpnam* places its result in that array and returns *s*.

*tmpnam* allows the user to control the choice of a directory. The argument *dir* points to the name of the directory in which the file is to be created. If *dir* is NULL or points to a string that is not a name for an appropriate directory, the path-prefix defined as **P\_tmpdir** in the `<stdio.h>` header file is used. If that directory is not accessible, **/tmp** will be used as a last resort. This entire sequence can be up-staged by providing an environment variable **TMPDIR** in the user's environment, whose value is the name of the desired temporary-file directory.

Many applications prefer their temporary files to have certain favorite initial letter sequences in their names. Use the *pfx* argument for this. This argument may be NULL or point to a string of up to five characters to be used as the first few characters of the temporary-file name.

*tmpnam* uses *malloc*(S) to get space for the constructed file name and returns a pointer to this area. Thus, any pointer value returned from *tmpnam* may serve as an argument to *free* (see *malloc*(S)). If *tmpnam* cannot return the expected result for any reason, i.e., *malloc*(S) failed, or none of the above mentioned attempts to find an appropriate directory was successful, a NULL pointer will be returned.



## See Also

---

`creat(S)`, `unlink(S)`, `fopen(S)`, `malloc(S)`, `mktemp(S)`, `tmpfile(S)`

## Notes

---

These functions generate a different file name each time they are called.

Files created using these functions and either *fopen(S)* or *creat(S)* are temporary only in the sense that they reside in a directory intended for temporary use, and their names are unique. It is the user's responsibility to use *unlink(S)* to remove the file when its use is ended.

## Notes

---

If called more than 17,576 times in a single process, these functions will start recycling previously used names.

Between the time a file name is created and the file is opened, it is possible for some other process to create a file with the same name. This can never happen if that other process is using these functions or *mktemp*, and the file names are chosen to render duplication by other means unlikely.

## Standards Conformance

---

*tmpnam* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

## trig: sin, cos, tan, asin, acos, atan, atan2

---

trigonometric functions

### Syntax

---

```
#include <math.h>
```

```
double sin (x)
double x;
```

```
double cos (x)
double x;
```

```
double tan (x)
double x;
```

```
double asin (x)
double x;
```

```
double acos (x)
double x;
```

```
double atan (x)
double x;
```

```
double atan2 (y, x)
double y, x;
```

### Description

---

The *sin*, *cos*, and *tan* functions return respectively the sine, cosine, and tangent of their argument,  $x$ , measured in radians.

*asin* returns the arcsine of  $x$ , in the range  $[-\pi/2, \pi/2]$ .

*acos* returns the arccosine of  $x$ , in the range  $[0, \pi]$ .

*atan* returns the arctangent of  $x$ , in the range  $[-\pi/2, \pi/2]$ .

*atan2* returns the arctangent of  $y/x$ , in the range  $(-\pi, \pi]$ , using the signs of both arguments to determine the quadrant of the return value.

### See Also

---

matherr(S)

## Diagnostics

---

*sin*, *cos*, and *tan* lose accuracy when their argument is far from zero. For arguments sufficiently large, these functions return zero when there would otherwise be a complete loss of significance. In this case a message indicating TLOSS error is printed on the standard error output. For less extreme arguments causing partial loss of significance, a PLOSS error is generated but no message is printed. In both cases, *errno* is set to **ERANGE**.

If the magnitude of the argument of *asin* or *acos* is greater than one, or if both arguments of *atan2* are zero, zero is returned and *errno* is set to **EDOM**. In addition, a message indicating DOMAIN error is printed on the standard error output.

These error-handling procedures may be changed with the function *matherr*(S).

## Standards Conformance

---

*acos*, *asin*, *atan*, *atan2*, *cos*, *sin* and *tan* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;  
and NIST FIPS 151-1.



## **tsearch, tfind, tdelete, twalk**

---

manage binary search trees

### **Syntax**

---

```
#include <search.h>
```

```
void *tsearch ((void *) key, (char **) rootp, compar)
int (*compar)();
```

```
void *tfind ((void *) key, (char **) rootp, compar)
int (*compar)();
```

```
void *tdelete ((void *) key, (char **) rootp, compar)
int (*compar)();
```

```
void twalk ((char *) root, action)
void (*action)();
```

### **Description**

---

The *tsearch*, *tfind*, *tdelete*, and *twalk* functions are routines for manipulating binary search trees. They are generalized from Knuth (6.2.2) Algorithms T and D. All comparisons are done with a user-supplied routine. This routine is called with two arguments, the pointers to the elements being compared. It returns an integer less than, equal to, or greater than 0, according to whether the first argument is to be considered less than, equal to, or greater than the second argument. The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

The *tsearch* function is used to build and access the tree. **key** is a pointer to a datum to be accessed or stored. If there is a datum in the tree equal to \*key (the value pointed to by key), a pointer to this found datum is returned. Otherwise, \*key is inserted, and a pointer to it returned. Only pointers are copied, so the calling routine must store the data. **rootp** points to a variable that points to the root of the tree. A NULL value for the variable pointed to by **rootp** denotes an empty tree; in this case, the variable will be set to point to the datum which will be at the root of the new tree.

Like *tsearch*, *tfind* will search for a datum in the tree, returning a pointer to it if found. However, if it is not found, *tfind* will return a NULL pointer. The arguments for *tfind* are the same as for *tsearch*.

*tdelete* deletes a node from a binary search tree. The arguments are the same as for *tsearch*. The variable pointed to by **rootp** will be changed if the deleted node was the root of the tree. *tdelete* returns a pointer to the parent of the deleted node, or a NULL pointer if the node is not found.

*twalk* traverses a binary search tree. **root** is the root of the tree to be traversed. (Any node in a tree may be used as the root for a walk below that node.) *Action* is the name of a routine to be invoked at each node. This routine is, in turn, called with three arguments. The first argument is the address of the node being visited. The second argument is a value from an enumeration data type *typedef enum { preorder, postorder, endorder, leaf } VISIT;* (defined in the `<search.h>` header file), depending on whether this is the first, second, or third time that the node has been visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a leaf. The third argument is the level of the node in the tree, with the root being level zero.

The pointers to the key and the root of the tree should be of type pointer-to-element, and cast to type pointer-to-character. Similarly, although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

## Example

---

The following code reads in strings and stores structures containing a pointer to each string and a count of its length. It then walks the tree, printing out the stored strings and their lengths in alphabetical order.

```
#include <search.h>
#include <stdio.h>

struct node { /* pointers to these are stored in the tree */
 char *string;
 int length;
};

char string_space[10000]; /* space to store strings */
struct node nodes[500]; /* nodes to store */
struct node *root = NULL; /* this points to the root */

main ()
{
 char *strptr = string_space;
 struct node *nodeptr = nodes;
 void print_node(), twalk();
 int i = 0, node_compare();
```

```

while (gets(strptr) != NULL && i++ < 500) {
 /* set node */
 nodeptr->string = strptr;
 nodeptr->length = strlen(strptr);
 /* put node into the tree */
 (void) tsearch((char *)nodeptr, (char **) &root,
 node_compare);
 /* adjust pointers, so we don't overwrite tree */
 strptr += nodeptr->length + 1;
 nodeptr++;
}
twalk((char *)root, print_node);
}
/*
 This routine compares two nodes, based on an
 alphabetical ordering of the string field.
*/
int
node_compare(node1, node2)
char *node1, *node2;
{
 return strcmp(((struct node *)node1)->string,
 ((struct node *) node2)->string);
}
/*
 This routine prints out a node, the first time
 twalk encounters it.
*/
void
print_node(node, order, level)
char **node;
VISIT order;
int level;
{
 if (order == preorder || order == leaf) {
 (void)printf("string = %20s, length = %d\n",
 *((struct node **)node)->string,
 *((struct node **)node)->length);
 }
}

```

## See Also

---

bsearch(S), hsearch(S), lsearch(S)

## Diagnostics

---

A NULL pointer is returned by *tsearch* if there is not enough space available to create a new node.

A NULL pointer is returned by *tfind* and *tdelete* if **rootp** is NULL on entry.



If the datum is found, both *tsearch* and *tfind* return a pointer to it. If not, *tfind* returns NULL, and *tsearch* returns a pointer to the inserted item.

## Warnings

---

The **root** argument to *twalk* is one level of indirection less than the **rootp** arguments to *tsearch* and *tdelete*.

There are two nomenclatures used to refer to the order in which tree nodes are visited. The *tsearch* function uses preorder, postorder, and endorder to respectively refer to visiting a node before any of its children, after its left child and before its right, and after both its children. The alternate nomenclature uses preorder, inorder, and postorder to refer to the same visits, which could result in some confusion over the meaning of postorder.

## Note

---

If the calling function alters the pointer to the root, results are unpredictable.

## Standards Conformance

---

*tdelete*, *tfind*, *tsearch* and *twalk* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## ttyname, isatty

---

find name of a terminal

### Syntax

---

```
char *ttyname (fildes)
int fildes;
```

```
int isatty (fildes)
int fildes;
```

### Description

---

The *ttyname* function returns a pointer to a string containing the null-terminated path name of the terminal device associated with file descriptor *fildes*.

*isatty* returns 1 if *fildes* is associated with a terminal device, 0 otherwise.

### Files

---

/dev/\*

### Diagnostics

---

The *ttyname* function returns a NULL pointer if *fildes* does not describe a terminal device in directory /dev .

### Note

---

The return value points to static data whose content is overwritten by each call.

### Standards Conformance

---

*isatty* and *ttyname* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

## ttyslot

---

find the slot in the utmp file of the current user

### Syntax

---

int `ttyslot` ( )

### Description

---

The *ttyslot* function returns the index of the current user's entry in the `/etc/utmp` file. This is accomplished by actually scanning the file `/etc/inittab` for the name of the terminal associated with the standard input, the standard output, or the error output (0, 1 or 2).

### Files

---

`/etc/inittab`  
`/etc/utmp`

### See Also

---

`getut(S)`, `ttynam(S)`

### Diagnostics

---

A value of 0 is returned if an error was encountered while searching for the terminal name or if none of the above file descriptors is associated with a terminal device.

### Standards Conformance

---

*ttyslot* is conformant with:

The X/Open Portability Guide II of January 1987.



# uadmin

---

administrative control

## Syntax

---

```
#include <sys/uadmin.h>
```

```
int uadmin (cmd, fcn, mdep)
int cmd, fcn, mdep;
```

## Description

---

The *uadmin* system call provides control for basic administrative functions. This system call is tightly coupled to the system administrative procedures and is not intended for general use. The argument *mdep* is provided for machine-dependent use and is not defined here.

As specified by *cmd*, the following commands are available:

**A\_SHUTDOWN** The system is shutdown. All user processes are killed, the buffer cache is flushed, and the root file system is unmounted. The action to be taken after the system has been shut down is specified by *fcn*. The functions are generic; the hardware capabilities vary on specific machines.

**AD\_HALT** Halt the processor until a key is entered to reboot the system.

**AD\_BOOT** Interactive reboot; user is prompted for system name.

**AD\_IBOOT** Interactive reboot; user is prompted for system name.

**AD\_PWRDOWN** Halt the processor; system remains down with no reboot option given.

**A\_REBOOT** The system stops immediately without any further processing. The action to be taken next is specified by *fcn* as above.

**A\_REMOUNT** The root file system is mounted again after having been fixed. This should be used only during the startup process.

A\_SETCONFIG Sets the system configuration. Currently, only the following function is available:

AD\_PANICBOOT If *mdep* is 1, this function causes the machine to reboot automatically after a kernel panic. If *mdep* is 0, this function causes the machine to wait for user intervention at the console before rebooting after a kernel panic.

The *uadmin* system call fails if any of the following is true:

[EPERM] The effective user ID is not super-user.

## Diagnostics

---

Upon successful completion, the value returned depends on *cmd* as follows:

|             |                  |
|-------------|------------------|
| A_SHUTDOWN  | No return value. |
| A_REBOOT    | No return value. |
| A_REMOUNT   | 0                |
| A_SETCONFIG | 0                |

Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

# ulimit

---

## get and set user limits

### Syntax

---

```
long ulimit (cmd, newlimit)
int cmd;
long newlimit;
```

### Description

---

This function provides for control over process limits. The *cmd* values available are:

- 1 Get the regular file size limit of the process. The limit is in units of 512-byte blocks and is inherited by child processes. Files of any size can be read.
- 2 Set the regular file size limit of the process to the value of *newlimit*. Any process may decrease this limit, but only a process with an effective user ID of super-user may increase the limit. *ulimit* fails and the limit is unchanged if a process with an effective user ID other than super-user attempts to increase its regular file size limit. [EPERM]
- 3 Get the maximum possible break value (see *brk(S)*).
- 4 Return configured value of NOFiles, the value for the maximum number of open files per process.

### See Also

---

*brk(S)*, *write(S)*

### Diagnostics

---

Upon successful completion, a non-negative value is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

### Warning

---

*ulimit* is effective in limiting the growth of regular files. Pipes are currently limited to 5,120 bytes.



## **Standards Conformance**

---

*ulimit* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## umask

---

set and get file creation mask

### Syntax

---

```
int umask (cmask)
int cmask;
```

### Description

---

The *umask* system call sets the process's file mode creation mask to *cmask* and returns the previous value of the mask. Only the low-order 9 bits of *cmask* and the file mode creation mask are used.

### See Also

---

chmod(S), creat(S), mknod(S), open(S), mkdir(C), sh(C)

### Diagnostics

---

The previous value of the file mode creation mask is returned.

### Standards Conformance

---

*umask* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
The X/Open Portability Guide II of January 1987;  
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;  
and NIST FIPS 151-1.

# umount

---

unmount a file system

## Syntax

---

```
int umount (file)
char *file;
```

## Description

---

The *umount* system call requests that a previously mounted file system contained on the block special device or directory identified by *file* be unmounted. *file* is a pointer to a path name. After unmounting the file system, the directory upon which the file system was mounted reverts to its ordinary interpretation.

The *umount* system call may be invoked only by the super-user.

The *umount* system call will fail if one or more of the following is true:

|             |                                                                                               |
|-------------|-----------------------------------------------------------------------------------------------|
| [EPERM]     | The process's effective user ID is not super-user.                                            |
| [EINVAL]    | <i>file</i> does not exist.                                                                   |
| [ENOTBLK]   | <i>file</i> is not a block special device.                                                    |
| [EINVAL]    | <i>file</i> is not mounted.                                                                   |
| [EBUSY]     | A file on <i>file</i> is busy.                                                                |
| [EFAULT]    | <i>file</i> points to an illegal address.                                                     |
| [EREMOTE]   | <i>file</i> is remote.                                                                        |
| [ENOLINK]   | <i>file</i> is on a remote machine, and the link to that machine is no longer active.         |
| [EMULTIHOP] | Components of the path pointed to by <i>file</i> require hopping to multiple remote machines. |
| [ENOTDIR]   | A component of the path-prefix is not a directory.                                            |
| [ENOENT]    | The named file does not exist.                                                                |



## See Also

---

mount(S)

## Diagnostics

---

Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## Standards Conformance

---

*umount* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.

## uname

---

get name of current system

### Syntax

---

```
#include <sys/utsname.h>
```

```
int uname (name)
struct utsname *name;
```

### Description

---

The *uname* system call stores information identifying the current system in the structure pointed to by *name*.

The *uname* system call uses the structure defined in *<sys/utsname.h>* whose members are:

```
char sysname[9];
char nodename[9];
char release[9];
char version[9];
char machine[9];
```

The *uname* system call returns a null-terminated character string naming the current system in the character array *sysname*. Similarly, *nodename* contains the name that the system is known by on a communications network. *release* and *version* further identify the operating system. *machine* contains a standard name that identifies the hardware that the system is running on.

[EFAULT] *uname* will fail if *name* points to an invalid address.

### See Also

---

uname(C)

### Diagnostics

---

Upon successful completion, a non-negative value is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## Standards Conformance

---

*uname* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;

and NIST FIPS 151-1.



## ungetc

---

push character back into input stream

### Syntax

---

```
#include <stdio.h>
```

```
int ungetc (c, stream)
```

```
int c;
```

```
FILE *stream;
```

### Description

---

The *ungetc* function inserts the character *c* into the buffer associated with an input *stream*. That character, *c*, will be returned by the next *getc*(*S*) call on that *stream*. The *ungetc* function returns *c*, and leaves the file *stream* unchanged.

One character of pushback is guaranteed, provided something has already been read from the stream and the stream is actually buffered.

If *c* equals **EOF**, *ungetc* does nothing to the buffer and returns **EOF**.

The *fseek*(*S*) function erases all memory of inserted characters.

### See Also

---

*fseek*(*S*), *getc*(*S*), *setbuf*(*S*), *stdio*(*S*)

### Diagnostics

---

*ungetc* returns **EOF** if it cannot insert the character.

### Notes

---

When *stream* is *stdin*, one character may be pushed back onto the buffer without a previous read statement.

## Standards Conformance

---

*ungetc* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

ANSI X3.159-198X C Language Draft Standard, May 13, 1988;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;

and NIST FIPS 151-1.

## unlink

---

remove directory entry

### Syntax

---

```
int unlink (path)
char *path;
```

### Description

---

*unlink* removes the directory entry named by the path name pointed to by *path*.

The named file is unlinked unless one or more of the following is true:

- |             |                                                                                                          |
|-------------|----------------------------------------------------------------------------------------------------------|
| [ENOTDIR]   | A component of the path prefix is not a directory.                                                       |
| [ENOENT]    | The named file does not exist.                                                                           |
| [EACCES]    | Search permission is denied for a component of the path prefix.                                          |
| [EACCES]    | Write permission is denied on the directory containing the link to be removed.                           |
| [EPERM]     | The named file is a directory and the effective user ID of the process is not super-user.                |
| [EBUSY]     | The entry to be unlinked is the mount point for a mounted file system.                                   |
| [ETXTBSY]   | The entry to be unlinked is the last link to a pure procedure (shared text) file that is being executed. |
| [EROFS]     | The directory entry to be unlinked is part of a read-only file system.                                   |
| [EFAULT]    | <i>path</i> points outside the process's allocated address space.                                        |
| [EINTR]     | A signal was caught during the <i>unlink</i> system call.                                                |
| [ENOLINK]   | <i>path</i> points to a remote machine and the link to that machine is no longer active.                 |
| [EMULTIHOP] | Components of <i>path</i> require hopping to multiple remote machines.                                   |



A file will not be unlinked when all of the following conditions are true:

- the parent directory has the sticky bit set
- the file is not writable by the user
- the user does not own the parent directory
- the user does not own the file
- the user is not root

When all links to a file have been removed and no process has the file open, the space occupied by the file is freed and the file ceases to exist. If one or more processes have the file open when the last link is removed, the removal is postponed until all references to the file have been closed.

## See Also

---

`close(S)`, `link(S)`, `open(S)`, `rm(C)`

## Diagnostics

---

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## Standards Conformance

---

*unlink* is conformant with:

- AT&T SVID Issue 2, Select Code 307-127;
- The X/Open Portability Guide II of January 1987;
- IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;
- and NIST FIPS 151-1.

## ustat

get file system statistics

### Syntax

```
#include <sys/types.h>
#include <ustat.h>
```

```
int ustat (dev, buf)
dev_t dev;
struct ustat *buf;
```

### Description

The *ustat* system call returns information about a mounted file system. *dev* is a device number identifying a device containing a mounted file system. *buf* is a pointer to a *ustat* structure that includes the following elements:

|         |                                       |
|---------|---------------------------------------|
| daddr_t | f_tfree; /* Total free blocks */      |
| ino_t   | f_tinode; /* Number of free inodes */ |
| char    | f_fname[6]; /* Filsys name */         |
| char    | f_fpack[6]; /* Filsys pack name */    |

The last two fields, *f\_name* and *f\_fpack* may not have significant information on all systems, and, in that case, will contain the null character.

The *ustat* system call will fail if one or more of the following is true:

- |           |                                                                                     |
|-----------|-------------------------------------------------------------------------------------|
| [EINVAL]  | <i>dev</i> is not the device number of a device containing a mounted file system.   |
| [EFAULT]  | <i>buf</i> points outside the process's allocated address space.                    |
| [EINTR]   | A signal was caught during a <i>ustat</i> system call.                              |
| [ENOLINK] | <i>dev</i> is on a remote machine and the link to that machine is no longer active. |
| [ECOMM]   | <i>dev</i> is on a remote machine and the link to that machine is no longer active. |

### See Also

stat(S), statfs(S), fs(F).

## Diagnostics

---

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

## Standards Conformance

---

*ustat* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;  
and The X/Open Portability Guide II of January 1987.



## utime

---

set file access and modification times

### Syntax

---

```
#include <sys/types.h>
int utime (path, times)
char *path;
struct utimbuf *times;
```

### Description

---

*path* points to a path name naming a file. The *utime* system call sets the access and modification times of the named file.

If *times* is **NULL**, the access and modification times of the file are set to the current time. A process must be the owner of the file or have write permission to use *utime* in this manner.

If *times* is not **NULL**, *times* is interpreted as a pointer to a *utimbuf* structure and the access and modification times are set to the values contained in the designated structure. Only the owner of the file or the super-user may use *utime* this way.

The times in the following structure are measured in seconds since 00:00:00 Greenwich Mean Time (GMT), Jan. 1, 1970.

```
struct utimbuf {
 time_t actime; /* access time */
 time_t modtime; /* modification time */
};
```

The *utime* system call will fail if one or more of the following is true:

- |           |                                                                                                                                    |
|-----------|------------------------------------------------------------------------------------------------------------------------------------|
| [ENOENT]  | The named file does not exist.                                                                                                     |
| [ENOTDIR] | A component of the path prefix is not a directory.                                                                                 |
| [EACCES]  | Search permission is denied by a component of the path prefix.                                                                     |
| [EPERM]   | The effective user ID is not super-user and not the owner of the file, and <i>times</i> is not <b>NULL</b> .                       |
| [EACCES]  | The effective user ID is not super-user and not the owner of the file, and <i>times</i> is <b>NULL</b> and write access is denied. |

|             |                                                                                           |
|-------------|-------------------------------------------------------------------------------------------|
| [EROFS]     | The file system containing the file is mounted read-only.                                 |
| [EFAULT]    | <i>times</i> is not <b>NULL</b> and points outside the process's allocated address space. |
| [EFAULT]    | <i>path</i> points outside the process's allocated address space.                         |
| [EINTR]     | A signal was caught during the <i>utime</i> system call.                                  |
| [ENOLINK]   | <i>path</i> points to a remote machine, and the link to that machine is no longer active. |
| [EMULTIHOP] | Components of <i>path</i> require hopping to multiple remote machines.                    |

## See Also

---

stat(S)

## Diagnostics

---

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

## Standards Conformance

---

*utime* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

## varargs

---

variable argument list

### Synopsis

---

```
#include <varargs.h>

function(va_alist)
va_dcl
va_list pvar;
va_start(pvar);
f = va_arg(pvar, type);
va_end(pvar);
```

### Description

---

This set of macros provides a means of writing portable procedures that accept variable argument lists. Routines having variable argument lists (such as *printf(S)*) that do not use *varargs* are inherently nonportable, since different machines use different argument passing conventions.

*va\_alist* is used in a function header to denote a variable argument list.

*va\_dcl* is a declaration for *va\_alist*. Note that there is no semicolon after *va\_dcl*.

*va\_list* is a type which can be used for the variable *pvar*, which is used to traverse the list. One such variable must always be declared.

*va\_start(pvar)* is called to initialize *pvar* to the beginning of the list.

*va\_arg(pvar, type)* will return the next argument in the list pointed to by *pvar*. *type* is the type the argument is expected to be. Different types can be mixed but it is up to the routine to know what type of argument is expected since it cannot be determined at runtime.

*va\_end(pvar)* is used to finish up.

Multiple traversals, each bracketed by *va\_start* ... *va\_end*, are possible.



## Example

---

```
#include <stdio.h>
#include <varargs.h>

main()
{
 show(2, 3.1, "but", 4.1, "end");
 show(1, 5.9, "hello");
 show(4, 6.2, "oops", 5.3, "blah", 5.1, "lovely",
 2.3, "madrigal");
}

/*
 * the first argument is an int which tells how many pairs follow.
 * the pairs are doubles and character pointers
 *
 * remember that when variables are passed to functions
 * floats are promoted to doubles and chars to ints.
 */
show(n, va_alist)
int n;
va_dcl
{
 va_list ap;
 int i;
 double f;
 char *p;

 va_start(ap);
 for (i = 0; i < n; ++i) {
 f = va_arg(ap, double);
 p = va_arg(ap, char *);
 printf("%4.1f %s\n", f, p);
 }
 va_end(ap);
}
```

## Notes

---

It is up to the calling routine to determine how many arguments there are, since it is not possible to determine this from the stack frame. For example, *excel* passes a 0 to signal the end of the list. *printf* can tell how many arguments are supposed to be there by the format of the list.

## **Standards Conformance**

---

*varargs* is conformant with:

The X/Open Portability Guide II of January 1987.

*va\_arg*, *va\_end*, *va\_list*, and *va\_start* are conformant with:

ANSI X3.159-198X C Language Draft Standard, May 13, 1988.

## **vprintf, vfprintf, vsprintf**

---

print formatted output of a varargs argument list

### **Syntax**

---

```
#include <stdio.h>
#include <varargs.h>
```

```
int vprintf (format, ap)
const char *format;"
va_list ap;
```

```
int vfprintf (stream, format, ap)
FILE *stream;
const char *format;"
va_list ap;
```

```
int vsprintf (s, format, ap)
const char *s, *format;"
va_list ap;
```

### **Description**

---

The *vprintf*, *vfprintf*, and *vsprintf* functions are the same as *printf*, *fprintf*, and *sprintf* respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by *varargs*(S).

### **Example**

---

The following demonstrates the use of *vfprintf* to write an error routine.

```
#include <stdio.h>
#include <varargs.h>
...
/*
 * error should be called by:
 * error(function_name, format, arg1, arg2...); */

/*VARARGS*/
void
error(va_alist)
/* Note that the function_name and format arguments cannot be
 * separately declared because of the definition of varargs. */
```



```
va_dcl
{
 va_list args;
 char *fmt;

 va_start(args);
 /* print out name of function causing error */
 (void)fprintf(stderr, "ERROR in %s: ", va_arg(args, char *));
 fmt = va_arg(args, char *);
 /* print out remainder of message */
 (void)vfprintf(stderr, fmt, args);
 va_end(args);
 (void)abort();
}
```

## See Also

---

printf(S), varargs(S)

## Standards Conformance

---

*vfprintf*, *vprintf* and *vsprintf* are conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

and ANSI X3.159-198X C Language Draft Standard, May 13, 1988.

## wait, waitpid

---

wait for child process to stop or terminate

### Syntax

---

```
#include <sys/types.h>
#include <sys/wait.h>
```

```
int wait (stat_loc)
int *stat_loc;
int wait ((int *) 0)
```

```
pid_t waitpid (pid, stat_loc, options)
pid_t pid;
int *stat_loc;
int options;
```

### Description

---

The *wait* and *waitpid* system calls suspend the calling process until one of the immediate children terminates or until a child that is being traced stops, because it has hit a break point. The *wait* system call returns prematurely if a signal is received. If a child process stops or terminates prior to the call on *wait*; return is immediate.

The *waitpid()* function behaves identically to the *wait()* function, if the *pid* argument has a value of -1 and the *options* argument has a value of zero. Otherwise, its behavior is modified by the values of the *pid* and *options* arguments.

The *pid* argument specifies a set of child processes for which status is requested. The *waitpid()* only returns the status of a child process from this set.

If *pid* is equal to -1, status is requested for any child process. In this respect, *waitpid()* is then equivalent to *wait()*.

If *pid* is greater than zero, it specifies the process ID of a single child process for which status is requested.

If *pid* is equal to zero, status is requested for any child process whose process group ID is equal to that of the calling process.

If *pid* is less than -1, status is requested for any child process whose process group ID is equal to the absolute value of *pid*.

The *options* argument is constructed from the bitwise inclusive or of zero or more of the following flags, defined in the header `<sys/wait.h>`:

- WNOHANG**      The *waitpid()* function does not suspend execution of the calling process if status is not immediately available for one of the child processes specified by *pid*.
- WUNTRACED**    If the implementation supports job control, the status of any child processes specified by *pid* that are stopped, and whose status has not yet been reported since they stopped, are also reported to the requesting process.

If *wait()* or *waitpid()* return because the status of a child process is available, these functions return a value equal to the process ID of the child process.

In this case, if the value of the argument *stat\_loc* is not NULL, information is stored in the location pointed to by *stat\_loc*. If and only if the status returned is from a terminated child process that returned a value of zero from *main()* or passed a value of zero as the *status* argument to *\_exit()* or *exit()*, the value stored at the location pointed to by *stat\_loc* is zero. Regardless of its value, this information may be interpreted using the following macros, which are defined in `<sys/wait.h>` and evaluate to integral expressions; the *stat\_val* argument is the integer value pointed to by *stat\_loc*.

**WIFEXITED(*stat\_val*)**  
Evaluates to a non-zero value if status was returned for a child process that terminated normally.

**WEXITSTATUS(*stat\_val*)**  
If the value of **WIFEXITED(*stat\_val*)** is non-zero, this macro evaluates to the low-order 8 bits of the *status* argument that the child process passed to *\_exit()* or *exit()*, or the value the child process returned from *main()*.

**WIFSIGNALED(*stat\_val*)**  
Evaluates to a non-zero value if status was returned for a child process that terminated due to the receipt of a signal that was not caught (see *signal(S)*).

**WTERMSIG(*stat\_val*)**  
If the value of **WIFSIGNALED(*stat\_val*)** is non-zero, this macro evaluates to the number of the signal that caused the termination of the child process.



WIFSTOPPED(*stat\_val*)

Evaluates to a non-zero value if status was returned for a child process that is currently stopped.

WSTOPSIG(*stat\_val*)

If the value of WIFSTOPPED(*stat\_val*) is non-zero, this macro evaluates to the number of the signal that caused the child process to stop.

If the information stored at the location pointed to by *stat\_loc* was stored there by a call to the *waitpid()* function that specified the WUNTRACED flag, exactly one of the macros WIFEXITED(*stat\_loc*), WIFSIGNALED(*stat\_loc*), and WIFSTOPPED(*stat\_loc*) evaluates to a non-zero value. If the information stored at the location pointed to by *stat\_loc* was stored there by a call to the *waitpid()* function that did not specify the WUNTRACED flag or by a call to the *wait()* function, exactly one of the macros WIFEXITED(*\*stat\_loc*) and WIFSIGNALED(*\*stat\_loc*) evaluates to a non-zero number.

If *stat\_loc* is non-zero, 16 bits of information called status are stored in the low order 16 bits of the location pointed to by *stat\_loc*. *status* can be used to differentiate between stopped and terminated child processes and if the child process terminated, status identifies the cause of termination and passes useful information to the parent. This is accomplished in the following manner:

If the child process stopped, the high order 8 bits of status will contain the number of the signal that caused the process to stop, and the low order 8 bits will be set equal to 0177.

If the child process terminated due to an *exit* call, the low order 8 bits of status will be zero, and the high order 8 bits will contain the low order 8 bits of the argument that the child process passed to *exit* (see *exit(S)*).

If the child process terminated due to a signal, the high order 8 bits of status will be zero, and the low order 8 bits will contain the number of the signal that caused the termination. In addition, if the low order seventh bit (that is, bit 200) is set, a "core image" will have been produced (see *signal(S)*).

If a parent process terminates without waiting for its child processes to terminate, the parent process ID of each child process is set to 1. This means the initialization process inherits the child processes (see *intro(S)*).

## Return Value

---

If the *wait()* or *waitpid()* functions return because the status of a child process is available, these functions return a value equal to the process ID of the child process for which status is reported. If the *wait()* or *waitpid()* functions return due to the delivery of a signal to the calling process, a value of -1 is returned and *errno* is set to [EINTR]. If the *waitpid()* function was invoked with WNOHANG set in *options*, it has at least one child process specified by *pid* for which status is not available, and status is not available for any process specified by *pid*, a value of zero is returned. Otherwise, a value of -1 is returned, and *errno* is set to indicate the error.

The *wait()* function fails and returns -1 if one or more of the following is true:

- |          |                                                                                                                         |
|----------|-------------------------------------------------------------------------------------------------------------------------|
| [ECHILD] | The calling process has no existing unwaited-for child processes.                                                       |
| [EINTR]  | The function was interrupted by a signal. The value of the location pointed to by <i>stat_loc</i> is <i>undefined</i> . |

If any of the following conditions occur, the *waitpid()* function returns -1 and sets *errno* to the corresponding value:

- |          |                                                                                                                         |
|----------|-------------------------------------------------------------------------------------------------------------------------|
| [ECHILD] | The process or process group defined by <i>pid</i> does not exist or is not a child of the calling process.             |
| [EINTR]  | The function was interrupted by a signal. The value of the location pointed to by <i>stat_loc</i> is <i>undefined</i> . |
| [EINVAL] | The value of the <i>options</i> argument is not valid.                                                                  |

## See Also

---

*exec(S)*, *exit(S)*, *fork(S)*, *intro(S)*, *pause(S)*, *ptrace(S)*, *signal(S)*, *times(S)*

## Warning

---

The *wait* system call fails and its actions are undefined if *stat\_loc* points to an invalid address.

## **Standards Conformance**

---

*wait* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.

*waitpid* is conformant with:

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;

and NIST FIPS 151-1.



## waitsem, nbwaitsem

---

awaits and checks access to a resource governed by a semaphore

### Syntax

---

```
int waitsem(sem_num);
int sem_num;
```

```
int nbwaitsem(sem_num);
int sem_num;
```

### Description

---

*waitsem* gives the calling process access to the resource governed by the semaphore *sem\_num*. If the resource is in use by another process, *waitsem* will put the process to sleep until the resource becomes available; *nbwaitsem* will return the error ENAVAIL. *waitsem* and *nbwaitsem* are used in conjunction with *sigsem* to allow synchronization of processes wishing to access a resource. One or more processes may *waitsem* on the given semaphore and will be put to sleep until the process which currently has access to the resource issues *sigsem*. *sigsem* causes the process which is next in line on the semaphore's queue to be rescheduled for execution. The semaphore's queue is organized in first in first out (FIFO) order.

### System Compatibility

---

*waitsem* can only be used to synchronize semaphores created under UNIX Version 3.0, not for UNIX System V semaphores.

### See Also

---

creatsem(S), opensem(S), sigsem(S)

### Diagnostics

---

*waitsem* returns the value (int) -1 if an error occurs. If *sem\_num* has not been previously opened by a call to *opensem* or *creatsem*, *errno* is set to EBADF. If *sem\_num* does not refer to a semaphore type file, *errno* is set to ENOTNAM. All processes waiting (or attempting to wait) on the semaphore return with *errno* set to ENAVAIL when the process controlling the semaphore exits without relinquishing control

(thereby leaving the resource in an undeterminate state). If a process does two *waitsems* in a row without doing an intervening *sigsem*, *errno* is set to EINVAL.

## Notes

---

This feature is a XENIX specific enhancement and may not be present in all UNIX implementations. This routine must be linked with the linker option **-lx**.

## write

---

write on a file

### Syntax

---

```
int write (fildes, buf, nbyte)
int fildes;
char *buf;
unsigned nbyte;
```

### Description

---

*fildes* is a file descriptor obtained from a *creat*(S), *open*(S), *dup*(S), *fcntl*(S), or *pipe*(S) system call.

The *write* system call attempts to write *nbyte* bytes from the buffer pointed to by *buf* to the file associated with the *fildes*.

On devices capable of seeking, the actual writing of data proceeds from the position in the file indicated by the file pointer. Upon return from *write*, the file pointer is incremented by the number of bytes actually written.

On devices incapable of seeking, writing always takes place starting at the current position. The value of a file pointer associated with such a device is undefined.

If the *O\_PEND* flag of the file status flags is set, the file pointer will be set to the end of the file prior to each write.

For regular files, if the *O\_SYNC* flag of the file status flags is set, the write will not return until both the file data and file status have been physically updated. This function is for special applications that require extra reliability at the cost of performance. For block special files, if *O\_SYNC* is set, the write will not return until the data has been physically updated.

A write to a regular file will be blocked if mandatory file/record locking is set (see *chmod*(S)), and there is a record lock owned by another process on the segment of the file to be written. If *O\_NDELAY* is not set, the write will sleep until the blocking record lock is removed.

For STREAMS (see *intro*(S)) files, the operation of *write* is determined by the values of the minimum and maximum *nbyte* range ("packet size") accepted by the *stream*. These values are contained in the topmost *stream* module. Unless the user pushes the topmost module, these values cannot be set or tested from user level. If *nbyte* falls within the packet size range, *nbyte* bytes will be written. If *nbyte* does



not fall within the range and the minimum packet size value is zero, *write* will break the buffer into maximum packet size segments prior to sending the data downstream (the last segment may contain less than the maximum packet size). If *nbyte* does not fall within the range and the minimum value is non-zero, *write* will fail with *errno* set to ERANGE. Writing a zero-length buffer (*nbyte* is zero) sends zero bytes with zero returned.

For STREAMS files, if O\_NDELAY is not set and the *stream* cannot accept data (the *stream* write queue is full due to internal flow control conditions), *write* will block until data can be accepted. O\_NDELAY will prevent a process from blocking due to flow control conditions. If O\_NDELAY is set and the *stream* cannot accept data, *write* will fail. If O\_NDELAY is set and part of the buffer has been written when a condition in which the *stream* cannot accept additional data occurs, *write* will terminate and return the number of bytes written.

The *write* system call will fail and the file pointer will remain unchanged if one or more of the following is true:

- |           |                                                                                                                                  |
|-----------|----------------------------------------------------------------------------------------------------------------------------------|
| [EAGAIN]  | Mandatory file/record locking was set, O_NDELAY was set, and there was a blocking record lock.                                   |
| [EAGAIN]  | Total amount of system memory available when reading via raw IO is temporarily insufficient.                                     |
| [EAGAIN]  | Attempt to write to a <i>stream</i> that cannot accept data with the O_NDELAY flag set.                                          |
| [EBADF]   | <i>fdes</i> is not a valid file descriptor open for writing.                                                                     |
| [EDEADLK] | The write was going to go to sleep and cause a deadlock situation to occur.                                                      |
| [EFAULT]  | <i>buf</i> points outside the process's allocated address space.                                                                 |
| [EFBIG]   | An attempt was made to write a file that exceeds the process's file size limit or the maximum file size (see <i>ulimit</i> (S)). |
| [EINTR]   | A signal was caught during the <i>write</i> system call.                                                                         |
| [EINVAL]  | Attempt to write to a <i>stream</i> linked below a multiplexer.                                                                  |
| [ENOLCK]  | The system record lock table was full, so the write could not go to sleep until the blocking record lock was removed.            |

- [ENOLINK] *fildev* is on a remote machine and the link to that machine is no longer active.
- [ENOSPC] During a *write* to an ordinary file, there is no free space left on the device.
- [ENXIO] A hangup occurred on the *stream* being written to.
- [EPIPE and SIGPIPE signal] An attempt is made to write to a pipe that is not open for reading by any process.
- [ERANGE] Attempt to write to a *stream* with *nbyte* outside specified minimum and maximum write range, and the minimum value is non-zero.
- [EIO] A physical I/O error has occurred.

If a *write* requests that more bytes be written than there is room for (for example, the *ulimit* (see *ulimit*(S)) or the physical end of a medium), only as many bytes as there is room for will be written. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A write of 512-bytes will return 20. The next write of a non-zero number of bytes will give a failure return (except as noted below).

If the file being written is a pipe (or FIFO) and the *O\_NDELAY* flag of the file flag word is set, then write to a full pipe (or FIFO) will return a count of 0. Otherwise (*O\_NDELAY* clear), writes to a full pipe (or FIFO) will block until space becomes available.

A write to a STREAMS file can fail if an error message has been received at the stream head. In this case, *errno* is set to the value included in the error message.

## See Also

---

*creat*(S), *dup*(S), *fcntl*(S), *intro*(S), *lseek*(S), *open*(S), *pipe*(S), *ulimit*(S)

## Diagnostics

---

Upon successful completion the number of bytes actually written is returned. Otherwise, -1 is returned, and *errno* is set to indicate the error.

**Standards Conformance**

---

*write* is conformant with:

AT&T SVID Issue 2, Select Code 307-127;

The X/Open Portability Guide II of January 1987;

IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent  
System Support;

and NIST FIPS 151-1.



## xlist, fxlist

---

gets name list entries from files

### Syntax

---

```
#include <a.out.h>
```

```
int xlist(filename, xl)
char *filename;
struct xlist xl[];
```

```
#include <a.out.h>
#include <stdio.h>
int fxlist(fp, xl)
FILE *fp;
struct xlist xl[];
```

### Description

---

*fxlist* performs the same function as *xlist*, except that *fxlist* accepts a pointer to a previously opened file instead of a filename.

*xlist* examines the name list in the given executable output file and selectively extracts a list of values. The given executable file can be either UNIX format or COFF. The name list structure *xl* consists of an array of *xlist* structures containing names, types, values, and segment values (if applicable). The list is terminated by either a pointer to a null name or a null pointer. Each name is looked up in the name list of the file. If the name is found, the type and value of the name are inserted into the next two fields. The segment value (if it exists) is inserted in the third field. If the name is not found, both entries are set to zero. See *a.out*(F) for a discussion of the *xlist* structure.

*x.out* and *a.out* formats are understood, as well as 8086 relocatable and *x.out* segmented formats.

If the symbol table is in *a.out* format, and if the symbol name given to *xlist* is longer than eight characters, only the first eight characters are used for comparison. In all other cases, the name given to *xlist* must be the same length as a name list entry in order to match.

If two or more symbols happen to match the name given to *xlist*, then the type and value used will be those of the last symbol found.

## See Also

---

a.out(F)

## Diagnostics

---

*xlist* returns -1 and sets all type entries to zero if the file cannot be read, is not an object file, or contains an invalid name list. Otherwise, *xlist* returns zero. A return value of zero does *not* indicate that any or all of the given symbols were found.

|                                  |                                        |                 |
|----------------------------------|----------------------------------------|-----------------|
| acct: acctdisk, acctdusg,        | accton, acctwtm - overview of/         | acct(ADM)       |
| process accounting               | accton: turns on accounting . . .      | accton(ADM)     |
| accounting acctprc:              | acctprc: acctprc1, acctprc2 - . . .    | acctprc(ADM)    |
| acctprc: acctprc1,               | acctprc1, acctprc2 - process . . .     | acctprc(ADM)    |
| dodisk, lastlogin, monacct,/     | acctprc2 - process accounting . . .    | acctprc(ADM)    |
| acctdisk, acctdusg, accton,      | acctsh: chargefee, ckpacct, . . .      | acctsh(ADM)     |
| orderly release/ t_rcvrel:       | acctwtm - overview of/ acct: . . .     | acct(ADM)       |
| sin, cos, tan, asin,             | acknowledge receipt of an . . .        | t_rcvrel(NSL)   |
| initcond: special security       | acos, atan, atan2: performs/ . . .     | trig(S)         |
| /interface for audit subsystem   | actions for init and getty . . .       | initcond(ADM)   |
| killall: kill all                | activation, termination, . . .         | auditcmd(ADM)   |
| sag: system                      | active processes . . .                 | killall(ADM)    |
| sar, sa1, sa2, sadc - system     | activity graph . . .                   | sag(ADM)        |
| prints current SCCS file editing | activity report package sar: . . .     | sar(ADM)        |
| report process data and system   | activity sact: . . .                   | sact(CP)        |
| information about system         | activity timex: time a command; . . .  | timex(ADM)      |
| /vidunmap: supports video        | activity uptime: displays . . .        | uptime(C)       |
| debugger                         | adapter driver development . . .       | video(K)        |
| device driver/ idinstall:        | adb: invokes a general-purpose . . .   | adb(CP)         |
| acctmrg: merge or                | add, delete, update, or get . . .      | idinstall(ADM)  |
| /_nheapwalk: returns the         | add total accounting files . . .       | acctmrg(ADM)    |
| paddr: returns virtual           | address of the next heap entry/ . . .  | _fheapwalk(DOS) |
| /vasmapped, vasunbind: virtual   | address pointer to block data . . .    | paddr(K)        |
| vtop: convert a virtual          | address space memory routines . . .    | vas(K)          |
| t_bind: bind an                  | address to a physical address . . .    | vtop(K)         |
| db_write: transfers from a user  | address to a transport endpoint . . .  | t_bind(NSL)     |
| bytes to and from a physical     | address to contiguous memory . . .     | db_write(K)     |
| from physical memory to a user   | address copyio: copies . . .           | copyio(K)       |
| converts virtual and physical    | address db_read: transfers data . . .  | db_read(K)      |
| get the offset and segment of an | addresses ptok, ktok: . . .            | ptok(K)         |
| from or writes a byte to an I/O  | address FP_OFF, FP_SEG: . . .          | fp_off(DOS)     |
| copies bytes from a specific     | address inb, outb: reads a byte . . .  | inb(K)          |
| a 32-bit word to a physical I/O  | address movedata: . . .                | movedata(DOS)   |
| word from or to a physical I/O   | address /outd: reads, writes . . .     | ind(K)          |
| a virtual address to a physical  | address /reads, writes a 16-bit . . .  | inw(K)          |
| device's queue diskort:          | address vtop: convert . . .            | vtop(K)         |
| nl:                              | adds a block I/O request to a . . .    | diskort(K)      |
| lineprinters lpinit:             | adds line numbers to a file . . .      | nl(C)           |
| swapadd:                         | adds, reconfigures and maintains . . . | lpinit(ADM)     |
| putenv: changes or               | adds swap area . . .                   | swapadd(S)      |
| SCCS files                       | adds value to environment . . .        | putenv(S)       |
| IP print service lpfilter:       | admin: creates and administers . . .   | admin(CP)       |
| IP print service lpforms:        | administer filters used with the . . . | lpfilter(ADM)   |
| admin: creates and               | administer forms used with the . . .   | lpforms(ADM)    |
| netutil:                         | administers SCCS files . . .           | admin(CP)       |
| uinstall:                        | administers the UNIX network . . .     | netutil(ADM)    |
| /Menu driven at and cron         | administers UUCP control files . . .   | uinstall(ADM)   |
| auditsh: menu driven audit       | administration utility . . .           | atcronsh(ADM)   |
| backupsh: menu driven backup     | administration utility . . .           | auditsh(ADM)    |
| sysadmsh: menu driven system     | administration utility . . .           | backupsh(ADM)   |
|                                  | administration utility . . .           | sysadmsh(ADM)   |



|                                  |                                  |                |
|----------------------------------|----------------------------------|----------------|
| menu driven lp print service     | administration utility lpsh:     | lpsh(ADM)      |
| network listener service         | administration nlsadmin:         | nlsadmin(ADM)  |
| uadmin:                          | administrative control           | uadmin(ADM)    |
| uadmin:                          | administrative control           | uadmin(S)      |
| swap: swap                       | administrative interface         | swap(ADM)      |
| authorization/ authtsh:          | administrator interface for      | authtsh(ADM)   |
| alarm: sets a process' clock     | alarm clock                      | alarm(S)       |
|                                  | alarm: sets a process' alarm     | alarm(S)       |
| /MMDF hashed database of         | alias and routing information    | dbmbuild(ADM)  |
| mmdfalias: converts XENIX-style  | aliases file to MMDF/            | mmdfalias(ADM) |
| program's stack                  | alloca: allocates bytes from the | alloca(DOS)    |
| t_alloc:                         | allocate a library structure     | t_alloc(NSL)   |
| malloc, _fmalloc, _nmalloc:      | allocate main memory             | malloc(DOS)    |
| dma_relse: releases previously   | allocated DMA channel            | dma_relse(K)   |
| changes the size of a previously | allocated memory block _expand:  | _expand(DOS)   |
| _fmsize, _nmsize: return size of | allocated memory block _msize,   | _msize(DOS)    |
| /releases memory previously      | allocated with sptalloc          | sptfree(K)     |
| dma_alloc:                       | allocates a DMA channel          | dma_alloc(K)   |
| halloc:                          | allocates a huge array           | halloc(DOS)    |
| contiguous/ db_alloc, db_free:   | allocates and frees physically   | db_alloc(K)    |
| program's stack                  | allocates bytes from the         | alloca(DOS)    |
| initialization memget:           | allocates contiguous memory at   | memget(K)      |
| brkctl:                          | allocates data in a far segment  | brkctl(S)      |
| malloc, free, realloc, calloc:   | allocates main memory            | malloc(S)      |
| maps a device into/ sptalloc:    | allocates temporary memory or    | sptalloc(K)    |
| sbrk: change data segment space  | allocation brk,                  | brk(S)         |
| brk: changes data segment space  | allocation sbrk,                 | sbrk(S)        |
| buffering and size setvbuf:      | allow user control over stream   | setvbuf(DOS)   |
| file inittab:                    | alternative login terminals      | inittab(F)     |
| reduce: perform audit data       | analysis and reduction           | reduce(ADM)    |
| generates programs for lexical   | analysis lex:                    | lex(CP)        |
| temporarily auths: print         | and/or restrict authorizations   | auths(C)       |
| link editor output               | a.out: format of assembler and   | a.out(F)       |
|                                  | ar: archive file format          | ar(F)          |
| libraries                        | ar: maintains archives and       | ar(CP)         |
| libraries                        | ar: maintains archives and       | ar(XNX)        |
| dc: invokes an                   | arbitrary precision calculator   | dc(C)          |
| cpio: format of cpio             | archive                          | cpio(F)        |
| pax: portable                    | archive exchange                 | pax(C)         |
| ar:                              | archive file format              | ar(F)          |
| archive header of a member of an | archive file ldahread: read the  | ldahread(S)    |
| convert: convert                 | archive files to common formats  | convert(CP)    |
| tar:                             | archive format                   | tar(F)         |
| archive file ldahread: read the  | archive header of a member of an | ldahread(S)    |
| ar: maintains                    | archives and libraries           | ar(CP)         |
| ar: maintains                    | archives and libraries           | ar(XNX)        |
| tar:                             | archives files                   | tar(C)         |
| cpio: copies file                | archives in and out              | cpio(C)        |
| ranlib: converts                 | archives to random libraries     | ranlib(CP)     |
| the names of files on a backup   | archive xdumpdir: prints         | xdumpdir(C)    |
| swapadd: adds swap               | area                             | swapadd(S)     |

|                                  |                                       |                   |
|----------------------------------|---------------------------------------|-------------------|
| varargs: variable                | argument list . . . . .               | varargs(S)        |
| output of a varargs              | argument list /Prints formatted . .   | vprintf(S)        |
| getopt: gets option letter from  | argument vector . . . . .             | getopt(S)         |
| expr: evaluates                  | arguments as an expression . . .      | expr(C)           |
| echo: echoes                     | arguments . . . . .                   | echo(C)           |
| malloc: allocates a huge         | array . . . . .                       | malloc(DOS)       |
| lsearch: performs a linear       | array search lfind, . . . . .         | lfind(DOS)        |
| ascii: map of the                | aSCII character set . . . . .         | ascii(M)          |
| character set                    | ascii: map of the ASCII . . . . .     | ascii(M)          |
| atof, atoi, atol: converts       | aSCII to numbers . . . . .            | atof(S)           |
| between long integer and base 64 | aSCII a64l, l64a: converts . . . .    | a64l(S)           |
| tzset: converts date and time to | aSCII /gmtime, asctime, . . . .       | ctime(S)          |
| and/ ctime, localtime, gmtime,   | asctime, tzset: converts date . . .   | ctime(S)          |
| performs/ sin, cos, tan,         | asin, acos, atan, atan2: . . . .      | trig(S)           |
| commands help:                   | asks for help about SCCS . . . .      | help(CP)          |
| time of day                      | asktime: prompts for the correct .    | asktime(ADM)      |
| output a.out: format of          | assembler and link editor . . . .     | a.out(F)          |
| as: common                       | assembler . . . . .                   | as(CP)            |
| asx: XENIX 8086/186/286/386      | assembler . . . . .                   | asx(CP)           |
| masm: invokes the XENIX          | assembler . . . . .                   | masm(CP)          |
| program                          | assert: helps verify validity of . .  | assert(S)         |
| deassigns devices                | assign, deassign: assigns and . . .   | assign(C)         |
| /copydvagent: manipulate device  | assignment database entry . . . .     | getdvagent(S)     |
| freopen:                         | assigns a new file to a stream . . .  | freopen(DOS)      |
| assign, deassign:                | assigns and deassigns devices . . .   | assign(C)         |
| setbuf, setvbuf:                 | assigns buffering to a stream . . .   | setbuf(S)         |
| setkey:                          | assigns the function keys . . . .     | setkey(C)         |
| close the event queue and all    | associated devices ev_close: . . . .  | ev_close(S)       |
| getpriv: get system privileges   | associated with this process . . . .  | getpriv(S)        |
| assembler                        | asx: XENIX 8086/186/286/386 . . .     | asx(CP)           |
| a later time                     | at, batch: executes commands at . .   | at(C)             |
| sin, cos, tan, asin, acos,       | atan, atan2: performs/ . . . . .      | trig(S)           |
| sin, cos, tan, asin, acos, atan, | atan2: performs trigonometric/ . .    | trig(S)           |
| cron administration utility      | atcronsh: menu driven at and . . . .  | atcronsh(ADM)     |
| termination                      | atexit: calls a process at . . . . .  | atexit(DOS)       |
| termination                      | atexit: calls a process at . . . . .  | atexit(S)         |
| to numbers                       | atof, atoi, atol: converts ASCII . .  | atof(S)           |
| double-precision/ strtod,        | atof: converts a string to a . . . .  | strtod(S)         |
| numbers atof,                    | atoi, atol: converts ASCII to . . .   | atof(S)           |
| integer strtol, atol,            | atoi: converts string to . . . . .    | strtol(S)         |
| integer strtol,                  | atol, atoi: converts string to . . .  | strtol(S)         |
| atof, atoi,                      | atol: converts ASCII to numbers . .   | atof(S)           |
| rcc:                             | AT&T C compiler . . . . .             | rcc(CP)           |
| QIC-24/QIC-02 tape/ tapecntl:    | AT&T tape control for . . . . .       | tapecntl(C)       |
| filesystem backup/ restore:      | AT&T UNIX incremental . . . . .       | restore(ADM)      |
| xt: multiplexed tty driver for   | AT&T windowing terminals . . . .      | xt(HW)            |
| /Print file to printer           | attached to a serial console . . . .  | consoleprint(ADM) |
| lprint: print to a printer       | attached to the user's terminal . . . | lprint(C)         |
| data segment sdget, sdfree:      | attaches and detaches a shared . . .  | sdget(S)          |
| tunable parameter idtune:        | attempts to set value of a . . . .    | idtune(ADM)       |
| discr: check discretionary       | attributes of files and programs . .  | discr(S)          |



|                                  |                                        |                 |
|----------------------------------|----------------------------------------|-----------------|
| auditsh: menu driven             | audit administration utility . . . .   | auditsh(ADM)    |
| device                           | audit: audit subsystem interface . . . | audit(ADM)      |
| by the audit/ auditd: read       | audit collection files generated . . . | auditd(ADM)     |
| reduction reduce: perform        | audit data analysis and . . . . .      | reduce(ADM)     |
| authaudit: produce               | audit records due to/ . . . . .        | authaudit(S)    |
| events dlvr_audit: produce       | audit records for subsystem . . . .    | dlvr_audit(ADM) |
| /audit_close: open and access    | audit session data on a record/ . . .  | audit(S)        |
| auditcmd: command interface for  | audit subsystem activation,/ . . . .   | auditcmd(ADM)   |
| files generated by the           | audit subsystem and /collection . .    | auditd(ADM)     |
| audit:                           | audit subsystem interface device . .   | audit(ADM)      |
| audit/ audit_open, audit_read,   | audit_close: open and access . . . .   | audit(S)        |
| audit subsystem activation,/     | auditcmd: command interface for . .    | auditcmd(ADM)   |
| files generated by the audit/    | auditd: read audit collection . . . .  | auditd(ADM)     |
| chg_audit: enables and disable   | auditing for the next session . . . .  | chg_audit(ADM)  |
| audit_close: open and access/    | audit_open, audit_read, . . . . .      | audit(S)        |
| and access audit/ audit_open,    | audit_read, audit_close: open . . . .  | audit(S)        |
| administration utility           | auditsh: menu driven audit . . . . .   | auditsh(ADM)    |
| due to authentication events     | authaudit: produce audit records . .   | authaudit(S)    |
|                                  | authcap: authentication database . .   | authcap(ADM)    |
| the authentication database      | authcap: get information from . . . .  | authcap(S)      |
| consistency of Authentication/   | authck: check internal . . . . .       | authck(ADM)     |
| authcap:                         | authentication database . . . . .      | authcap(ADM)    |
| /get information from the        | authentication database . . . . .      | authcap(S)      |
| dblock: lock the entire          | Authentication database . . . . .      | dblock(S)       |
| /examine system files against    | authentication database . . . . .      | integrity(ADM)  |
| check internal consistency of    | authentication database authck: . .    | authck(ADM)     |
| return status based on fields of | authentication database fields: . .    | fields(S)       |
| /produce audit records due to    | authentication events . . . . .        | authaudit(S)    |
| /administrator interface for     | authorization subsystem . . . . .      | authsh(ADM)     |
| auths: print and/or restrict     | authorizations temporarily . . . . .   | auths(C)        |
| authorizations temporarily       | auths: print and/or restrict . . . . . | auths(C)        |
| for authorization subsystem      | authsh: administrator interface . . .  | authsh(ADM)     |
| the system                       | autoboot: automatically boots . . . .  | autoboot(ADM)   |
| schedule: database for           | automated system backups . . . . .     | schedule(ADM)   |
| autoboot:                        | automatically boots the system . . .   | autoboot(ADM)   |
| resource/ waitsem, nbwaitsem:    | awaits and checks access to a . . . .  | waitsem(S)      |
| processes wait:                  | awaits completion of background . .    | wait(C)         |
| a pattern in a file              | awk: searches for and processes . . .  | awk(C)          |
| mvdevice: video driver           | backend configuration file . . . . .   | mvdevice(F)     |
| wait: awaits completion of       | background processes . . . . .         | wait(C)         |
| backupsh: menu driven            | backup administration utility . . . .  | backupsh(ADM)   |
| prints the names of files on a   | backup archive xdumpdir: . . . . .     | xdumpdir(C)     |
| ssdate: prints and sets          | backup dates . . . . .                 | ssdate(C)       |
| /Default                         | backup device information . . . . .    | archive(F)      |
| backup: performs UNIX            | backup functions . . . . .             | backup(ADM)     |
| filesystem backup                | backup: performs incremental . . . .   | backup(ADM)     |
| backup functions                 | backup: performs UNIX . . . . .        | backup(ADM)     |
| incremental filesystem           | backup restore /UNIX . . . . .         | restore(ADM)    |
| performs incremental filesystem  | backup backup: . . . . .               | backup(ADM)     |
| error-checking filesystem        | backup fsave: interactive, . . . . .   | fsave(ADM)      |
| incremental filesystem           | backup /Performs XENIX . . . . .       | xbackup(ADM)    |



|                                  |                                           |                   |
|----------------------------------|-------------------------------------------|-------------------|
| periodic semi-automated system   | backups fsphoto: performs . . .           | fsphoto(ADM)      |
| administration utility           | backupsh: menu driven backup . . .        | backupsh(ADM)     |
| database for automated system    | backups schedule: . . . . .               | schedule(ADM)     |
| fixed disk for flaws and creates | bad track table badtrk: scans . . .       | badtrk(ADM)       |
| flaws and creates bad track/     | badtrk: scans fixed disk for . . .        | badtrk(ADM)       |
|                                  | banner: prints large letters . . .        | banner(C)         |
| between long integer and         | base 64 ASCII /l64a: converts . . .       | a64l(S)           |
| and sets the configuration data  | base cmos: displays . . . . .             | cmos(HW)          |
| and sets the configuration data  | base cmos: displays . . . . .             | cmos(HW-86)       |
| fields: return status            | based on fields of/ . . . . .             | fields(S)         |
| names from pathnames             | basename: removes directory . . .         | basename(C)       |
| terminal capability data         | base termcap: . . . . .                   | termcap(M)        |
| terminal capability data         | base "terminfo:" . . . . .                | terminfo(M)       |
| audit session data on a record   | basis /open and access . . . . .          | audit(S)          |
| later time at,                   | batch: executes commands at a . . .       | at(C)             |
| /cfsetispeed, cfsetospeed:       | baud rate functions . . . . .             | cfspeed(S)        |
|                                  | bc: invokes a calculator . . . . .        | bc(C)             |
| initialization/ brc: brc,        | bcheckrc - system . . . . .               | brc(ADM)          |
| space                            | bcopy: copies bytes in kernel . . .       | bcopy(K)          |
| for diff                         | bdiff: compares files too large . . .     | bdiff(C)          |
|                                  | bdos: invokes a DOS system call . . .     | bdos(DOS)         |
|                                  | cb: beautifies C programs . . . . .       | cb(CP)            |
|                                  | dma_enable: begins DMA transfer . . . . . | dma_enable(K)     |
| j0, j1, jn, y0, y1, yn: performs | bessel functions bessel, . . . . .        | bessel(S)         |
| performs Bessel functions        | bessel, j0, j1, jn, y0, y1, yn: . . .     | bessel(S)         |
|                                  | bfs: scans big files . . . . .            | bfs(C)            |
| /perform conversions between MS  | binary and IEEE formats . . . . .         | diecetomsbin(DOS) |
| mail uuencode: decode a          | binary file for transmission via . . .    | uuencode(C)       |
| mail uuencode: encode a          | binary file for transmission via . . .    | uuencode(C)       |
| fixhdr: changes executable       | binary file headers . . . . .             | fixhdr(C)         |
| selected parts of executable     | binary files hdr: displays . . . . .      | hdr(CP)           |
| conversions between IEEE and MS  | binary format /perform . . . . .          | fiecetomsbin(DOS) |
| fread, fwrite: performs buffered | binary input and output . . . . .         | fread(S)          |
| bsearch: performs a              | binary search . . . . .                   | bsearch(S)        |
| tfind, tdelete, twalk: manages   | binary search trees tsearch, . . .        | tsearch(S)        |
| creates an instance of a         | binary semaphore creatsem: . . .          | creatsem(S)       |
| endpoint t_bind:                 | bind an address to a transport . . .      | t_bind(NSL)       |
| removes symbols and relocation   | bits strip: . . . . .                     | strip(CP)         |
| removes symbols and relocation   | bits strip: . . . . .                     | strip(XNX)        |
| rmb: remove extra                | blank lines from a file . . . . .         | rmb(M)            |
| brlse: releases a                | block buffer . . . . .                    | brlse(K)          |
| getablk: gets a buffer from the  | block buffer pool geteblk, . . . . .      | geteblk(K)        |
| virtual address pointer to       | block data paddr: returns . . . . .       | paddr(K)          |
| physio, physck: raw I/O for      | block drivers . . . . .                   | physio(K)         |
| hfree: deallocates a memory      | block . . . . .                           | hfree(DOS)        |
| shutdn: flushes                  | block I/O and halts the CPU . . .         | shutdn(S)         |
| queue diskort: adds a            | block I/O request to a device's . . .     | diskort(K)        |
| cmchk: reports hard disk         | block size . . . . .                      | cmchk(C)          |
| sigprocmask: examine and change  | blocked signals . . . . .                 | sigprocmask(S)    |
| of a previously allocated memory | block _expand: changes the size . . .     | _expand(DOS)      |
| return size of allocated memory  | block _msize, _fmsize, _nmsize: . . .     | _msize(DOS)       |

|                                  |                                       |                |
|----------------------------------|---------------------------------------|----------------|
| splni, splpp, spltty, splx:      | block/permit interrupts /splhi, . .   | spl(K)         |
| df: report number of free disk   | blocks . . . . .                      | df(C)          |
| calculates checksum and counts   | blocks in a file sum: . . . . .       | sum(C)         |
| accepts a number of 512-byte     | blocks . . . . .                      | login(M)       |
| sizes DMA request into 512-byte  | blocks dma_breakup: . . . . .         | dma_breakup(K) |
| _nfree: deallocate memory        | blocks free, _ffree, . . . . .        | free(DOS)      |
| fdswap: swaps default            | boot floppy drive . . . . .           | fdswap(ADM)    |
| boot: UNIX                       | boot program . . . . .                | boot(HW)       |
|                                  | boot: UNIX boot program . . . . .     | boot(HW)       |
| autoboot: automatically          | boots the system . . . . .            | autoboot(ADM)  |
| initialization procedures brc:   | brc, bcheckrc - system . . . . .      | brc(ADM)       |
| initialization procedures        | brc: brc, bcheckrc - system . . . . . | brc(ADM)       |
| requests pio_breakup:            | breaks up programmed I/O . . . . .    | pio_breakup(K) |
|                                  | brelese: releases a block buffer . .  | brelese(K)     |
| allocation sbrk,                 | brk: changes data segment space .     | sbrk(S)        |
| space allocation                 | brk, sbrk: change data segment . .    | brk(S)         |
| segment                          | brkctl: allocates data in a far . . . | brkctl(S)      |
| search                           | bsearch: performs a binary . . . . .  | bsearch(S)     |
| between bytes and clicks/        | btoc, btoms, ctob: converts . . . . . | btoc(K)        |
| bytes and clicks (memory/ btoc,  | btoms, ctob: converts between . . .   | btoc(K)        |
| brelese: releases a block        | buffer . . . . .                      | brelese(K)     |
| pool geteblk, getabl: gets a     | buffer from the block buffer . . . .  | geteblk(K)     |
| gets a buffer from the block     | buffer pool geteblk, getabl: . . . .  | geteblk(K)     |
| output fread, fwrite: performs   | buffered binary input and . . . . .   | fread(S)       |
| stdio: performs standard         | buffered input and output . . . . .   | stdio(S)       |
| allow user control over stream   | buffering and size setvbuf: . . . .   | setvbuf(DOS)   |
| setbuf, setvbuf: assigns         | buffering to a stream . . . . .       | setbuf(S)      |
| memicmp: compares                | buffers byte-by-byte . . . . .        | memicmp(DOS)   |
| flushall: flushes all output     | buffers . . . . .                     | flushall(DOS)  |
| getc, getcbp, getcf: read clist  | buffers getc, . . . . .               | getc(K)        |
| a character to the console       | buffer ungetch: returns . . . . .     | ungetch(DOS)   |
| idbuild:                         | build new UNIX system kernel . . . .  | idbuild(ADM)   |
| kernel link_unix:                | builds a new UNIX system . . . . .    | link_unix(ADM) |
| mknod:                           | builds special files . . . . .        | mknod(C)       |
| database of alias and/ dbmbuild: | builds the MMDF hashed . . . . .      | dbmbuild(ADM)  |
| I/O address inb, outb: reads a   | byte from or writes a byte to an . .  | inb(K)         |
| inp: returns a                   | byte . . . . .                        | inp(DOS)       |
| reads a byte from or writes a    | byte to an I/O address /outb: . . . . | inb(K)         |
| outp: writes a                   | byte to an output port . . . . .      | outp(DOS)      |
| memicmp: compares buffers        | byte-by-byte . . . . .                | memicmp(DOS)   |
| /btoms, ctob: converts between   | bytes and clicks (memory pages) . .   | btoc(K)        |
| space copyin, copyout: copies    | bytes between user and kernel . . .   | copyin(K)      |
| movedata: copies                 | bytes from a specific address . . . . | movedata(DOS)  |
| alloca: allocates                | bytes from the program's stack . . .  | alloca(DOS)    |
| bcopy: copies                    | bytes in kernel space . . . . .       | bcopy(K)       |
| dma_resid: returns the number of | bytes not transferred during a/ . . . | dma_resid(K)   |
| swab: swaps                      | bytes . . . . .                       | swab(S)        |
| address copyio: copies           | bytes to and from a physical . . . .  | copyio(K)      |
| 0 (zero)                         | bzero: sets memory locations to . .   | bzero(K)       |
| cc: invokes the                  | C compiler . . . . .                  | cc(CP)         |
| rcc: AT&T                        | C compiler . . . . .                  | rcc(CP)        |



|                                  |                                  |                |
|----------------------------------|----------------------------------|----------------|
| cflow: generates                 | C flow graph                     | cflow(CP)      |
| cpp: the                         | C language preprocessor          | cpp(CP)        |
| lint: checks                     | C language usage and syntax      | lint(CP)       |
| cxref: generates                 | C program cross-reference        | cxref(CP)      |
| cscope: interactively examine a  | C program                        | cscope(CP)     |
| ctrace:                          | C program debugger               | ctrace(CP)     |
| cb: beautifies                   | C programs                       | cb(CP)         |
| xref: cross-references           | C programs                       | xref(CP)       |
| xstr: extracts strings from      | C programs                       | xstr(CP)       |
| object file list: produce        | C source listing from a common   | list(CP)       |
| an error message file from       | C source mkstr: creates          | mkstr(CP)      |
| value of a complex number        | cabs: calculates the absolute    | cabs(DOS)      |
| distance hypot,                  | cabs: determines Euclidean       | hypot(S)       |
|                                  | cal: prints a calendar           | cal(C)         |
| blocks in a file sum:            | calculates checksum and counts   | sum(C)         |
| a complex number cabs:           | calculates the absolute value of | cabs(DOS)      |
| bc: invokes a                    | calculator                       | bc(C)          |
| invokes an arbitrary precision   | calculator dc:                   | dc(C)          |
| cal: prints a                    | calendar                         | cal(C)         |
| service                          | calendar: invokes a reminder     | calendar(C)    |
| mktime: converts local time to   | calendar time                    | mktime(S)      |
| bdos: invokes a DOS system       | call                             | bdos(DOS)      |
| pointer from last routine        | call error /gets error message   | strerror(DOS)  |
| pointer from last routine        | call error /gets error message   | strerror(S)    |
| intdos: invokes a DOS system     | call                             | intdos(DOS)    |
| intdosx: invokes a DOS system    | call                             | intdosx(DOS)   |
| longjmp: ends current system     | call with error                  | longjmp(K)     |
| exit: terminates the             | calling process                  | exit(DOS)      |
| malloc, free, realloc,           | calloc: allocates main memory    | malloc(S)      |
| atexit:                          | calls a process at termination   | atexit(DOS)    |
| atexit:                          | calls a process at termination   | atexit(S)      |
| cu:                              | calls another UNIX system        | cu(C)          |
| data returned by stat system     | call stat:                       | stat(F)        |
| requests to lineprinter lp,      | cancel: send/cancel              | lp(C)          |
| lineprinter lp                   | cancel: send/cancel requests to  | lp(C)          |
| from tty device                  | canon: processes raw input data  | canon(K)       |
| termcap: terminal                | capability data base             | termcap(M)     |
| terminfo: terminal               | capability data base             | terminfo(M)    |
| description into a terminfo/     | captainfo: convert a termcap     | captainfo(ADM) |
| pnch: file format for            | card images                      | pnch(F)        |
| text editor (variant of ex for   | casual users) edit:              | edit(C)        |
| files                            | cat: concatenates and displays   | cat(C)         |
|                                  | cb: beautifies C programs        | cb(CP)         |
| gcc: create a front-end to the   | cc command                       | gcc(CP)        |
|                                  | cc: invokes the C compiler       | cc(CP)         |
|                                  | cd: changes working directory    | cd(C)          |
| commentary of an SCCS delta      | cdc: changes the delta           | cdc(CP)        |
| value, floor,/ floor, fabs,      | ceil, fmod: performs absolute    | floor(S)       |
| /Performs absolute value, floor, | ceiling and remainder functions  | floor(S)       |
| cfspeed,/ cfspeed:               | cfgetspeed, cfgetospeed,         | cfspeed(S)     |
| cfspeed: cfgetspeed,             | cfgetospeed, cfspeed,/           | cfspeed(S)     |



|                                  |                                            |               |
|----------------------------------|--------------------------------------------|---------------|
|                                  | cflow: generates C flow graph . . .        | cflow(CP)     |
| rate/ /cfgetispeed, cfgetospeed, | cfsetispeed, cfsetospeed: baud . . .       | cfspeed(S)    |
| /cfgetospeed, cfsetispeed,       | cfsetospeed: baud rate functions . . .     | cfspeed(S)    |
| cfgetospeed, cfsetispeed,/       | cfspeed: cfgetispeed, . . . . .            | cfspeed(S)    |
|                                  | cgets: gets a string . . . . .             | cgets(DOS)    |
| delta: makes a delta             | (change) to an SCCS file . . . .           | delta(CP)     |
| directory chdir:                 | changes current working . . . .            | chdir(DOS)    |
| allocation sbrk, brk:            | changes data segment space . . .           | sbrk(S)       |
| headers fixhdr:                  | changes executable binary file . .         | fixhdr(C)     |
|                                  | chmod: changes file permissions . . . .    | chmod(DOS)    |
|                                  | rename: changes filename . . . . .         | rename(S)     |
|                                  | chgrp: changes group ID . . . . .          | chgrp(C)      |
|                                  | chmod: changes mode of a file . . . .      | chmod(S)      |
| environment putenv:              | changes or adds value to . . . .           | putenv(S)     |
|                                  | chown: changes owner ID . . . . .          | chown(C)      |
|                                  | nice: changes priority of a process . .    | nice(S)       |
| command chroot:                  | changes root directory for . . . .         | chroot(ADM)   |
| modification dates of/ settime:  | changes the access and . . . . .           | settime(ADM)  |
| of a file or directory chmod:    | changes the access permissions . .         | chmod(C)      |
| an SCCS delta cdc:               | changes the delta commentary of .          | cdc(CP)       |
| file newform:                    | changes the format of a text . . .         | newform(C)    |
| system fsname: prints or         | changes the name of a file . . . .         | fsname(ADM)   |
| file chown:                      | changes the owner and group of a .         | chown(S)      |
| pointer lseek:                   | changes the position of a file . . .       | lseek(DOS)    |
|                                  | chroot: changes the root directory . . .   | chroot(S)     |
|                                  | chsize: changes the size of a file . . . . | chsize(S)     |
| allocated memory block _expand:  | changes the size of a previously . .       | _expand(DOS)  |
|                                  | chdir: changes the working directory . .   | chdir(S)      |
|                                  | cd: changes working directory . . . .      | cd(C)         |
| dma_alloc: allocates a DMA       | channel . . . . .                          | dma_alloc(K)  |
| emunmap: disables mapping on a   | channel . . . . .                          | emunmap(K)    |
| list: list processor             | channel for MMDF . . . . .                 | list(ADM)     |
| emdupmap: duplicates             | channel mapping . . . . .                  | emdupmap(K)   |
| previously allocated DMA         | channel dma_relse: releases . . . .        | dma_relse(K)  |
| process NIC database into        | channel/domain tables nictable: . .        | nictable(ADM) |
| xtproto: multiplexed             | channels protocol used by/ . . . .         | xtproto(M)    |
| stream ungetc: pushes            | character back into input . . . .          | ungetc(S)     |
| the kernel cpass, passc: passes  | character between user space and .         | cpass(K)      |
| isatty: checks for a             | character device . . . . .                 | isatty(DOS)   |
| ioctl: controls                  | character devices . . . . .                | ioctl(S)      |
| fgetc, fgetchar: gets a          | character from a stream . . . . .          | fgetc(DOS)    |
| fubyte: gets a                   | character from user data space . .         | fubyte(K)     |
| getch: gets a                    | character . . . . .                        | getch(DOS)    |
| getche: gets and echoes a        | character . . . . .                        | getche(DOS)   |
| subyte: stores a                 | character in user data space . . .         | subyte(K)     |
| getchar: gets one                | character of input . . . . .               | getchar(K)    |
| putchar: prints a                | character on the console . . . . .         | putchar(K)    |
| getc, getchar, fgetc, getw: gets | character or word from a stream .          | getc(S)       |
| /putchar, fputc, putw: puts a    | character or word on a stream . .          | putc(S)       |
| ascii: map of the ASCII          | character set . . . . .                    | ascii(M)      |
| trchan: translate                | character sets . . . . .                   | trchan(M)     |

|                                 |                                       |                |
|---------------------------------|---------------------------------------|----------------|
| fputc, fputchar: write a        | character to a stream . . . . .       | fputc(DOS)     |
| ungetch: returns a              | character to the console buffer . .   | ungetch(DOS)   |
| putch: writes a                 | character to the console . . . . .    | putch(DOS)     |
| displays/changes hard disk      | characteristics dparam: . . . . .     | dparam(ADM)    |
| memmove: copies                 | characters between objects . . . .    | memmove(DOS)   |
| memmove: copies                 | characters between objects . . . .    | memmove(S)     |
| strrev: reverses the order of   | characters in a string . . . . .      | strrev(DOS)    |
| charater strset: sets all       | characters in a string to one . . .   | strset(DOS)    |
| strnset: perform operations on  | characters in strings /strncpy, . .   | strncat(DOS)   |
| ltoa: converts long integers to | characters . . . . .                  | ltoa(DOS)      |
| strlwr: converts uppercase      | characters to lowercase . . . . .     | strlwr(DOS)    |
| strupr: converts lowercase      | characters to uppercase . . . . .     | strupr(DOS)    |
| tr: translates                  | characters . . . . .                  | tr(C)          |
| ultoa: converts numbers to      | characters . . . . .                  | ultoa(DOS)     |
| wc: counts lines, words and     | characters . . . . .                  | wc(C)          |
| tolower, toascii: translates    | characters conv, toupper, . . . .     | conv(S)        |
| toascii: classifies or converts | characters /tolower, toupper, . . .   | ctype(S)       |
| characters in a string to one   | charater strset: sets all . . . . .   | strset(DOS)    |
| lastlogin, monacct,/ acctsh:    | chargefee, ckpacct, dodisk, . . .     | acctsh(ADM)    |
| directory                       | chdir: changes current working . .    | chdir(DOS)     |
| directory                       | chdir: changes the working . . . .    | chdir(S)       |
| non-obviousness goodpw:         | check a password for . . . . .        | goodpw(ADM)    |
| of files and programs discr:    | check discretionary attributes . .    | discr(S)       |
| authentication database authck: | check internal consistency of . . .   | authck(ADM)    |
| performs a minimal consistency  | check on the heap /_nheapchk: . .     | _fheapchk(DOS) |
| permissions file uuchek:        | check the uucp directories and . .    | uuchek(ADM)    |
| start identity: get or          | check uids or gids from program . .   | identity(S)    |
| verify machine is suitable for/ | check_basic_data_structures: . . .    | check_data(S)  |
| processed by fsck               | checklist: list of filesystems . . .  | checklist(F)   |
| has been submitted but not/     | checkmail: checks for mail which . .  | checkmail(C)   |
| report generator                | checkque: MMDF queue status . . .     | checkque(ADM)  |
| waitsem, nbwaitsem: awaits and  | checks access to a resource/ . . . .  | waitsem(S)     |
| fsck:                           | checks and repairs filesystems . .    | fsck(ADM)      |
| syntax lint:                    | checks C language usage and . . . .   | lint(CP)       |
| isatty:                         | checks for a character device . . . . | isatty(DOS)    |
| submitted but not/ checkmail:   | checks for mail which has been . .    | checkmail(C)   |
| grpcheck:                       | checks group file . . . . .           | grpcheck(C)    |
| pwcheck:                        | checks password file . . . . .        | pwcheck(C)     |
| keystroke kbhit:                | checks the console for a . . . . .    | kbhit(DOS)     |
| to be read rdchk:               | checks to see if there is data . . .  | rdchk(S)       |
| file sum: calculates            | checksum and counts blocks in a . .   | sum(C)         |
| auditing for the next session   | chg_audit: enables and disable . .    | chg_audit(ADM) |
| times: gets process and         | chgrp: changes group ID . . . . .     | chgrp(C)       |
| terminate wait: waits for a     | child process times . . . . .         | times(S)       |
| with ptrace for tracing a       | child process to stop or . . . . .    | wait(S)        |
| /sets up a DMA controller       | child process /in conjunction . . .   | paccess(S)     |
| libraries tool                  | chip for DMA transfer . . . . .       | dma_param(K)   |
| permissions of a file or/       | chkshlib: compare shared . . . . .    | chkshlib(CP)   |
|                                 | chmod: changes file permissions . .   | chmod(DOS)     |
|                                 | chmod: changes mode of a file . . .   | chmod(S)       |
|                                 | chmod: changes the access . . . . .   | chmod(C)       |



|                                  |                                      |               |
|----------------------------------|--------------------------------------|---------------|
|                                  | chown: changes owner ID . . . .      | chown(C)      |
| group of a file                  | chown: changes the owner and . .     | chown(S)      |
| for command                      | chroot: changes root directory . .   | chroot(ADM)   |
| directory                        | chroot: changes the root . . . .     | chroot(S)     |
| table                            | chrtbl: create a ctype locale . .    | chrtbl(M)     |
| file                             | chsize: changes the size of a . .    | chsize(S)     |
| monacct,/ acctsh: chargefee,     | ckpacct, dodisk, lastlogin, . . .    | acctsh(ADM)   |
| tolower, toupper, toascii:       | classifies or converts /isascii, . . | ctype(S)      |
| in directories specified         | cleantmp: remove temporary files .   | cleantmp(ADM) |
| STREAMS error logger             | cleanup program strclean: . . . .    | strclean(ADM) |
| uuclean: uucp spool directory    | clean-up . . . . .                   | uuclean(ADM)  |
| getpasswd: read or               | clear a password . . . . .           | getpasswd(S)  |
|                                  | clear: clears a terminal screen . .  | clear(C)      |
| floating-point status word       | _clear87: gets and clears the . . .  | _clear87(DOS) |
| stream status ferror, feof,      | clearerr, fileno: determines . . .   | ferror(S)     |
| clear:                           | clears a terminal screen . . . .     | clear(C)      |
| clri:                            | clears inode . . . . .               | clri(ADM)     |
| word _clear87: gets and          | clears the floating-point status . . | _clear87(DOS) |
| ctob: converts between bytes and | clicks (memory pages) /btoms, . .    | btoc(K)       |
| a shell command interpreter with | c-like syntax csh: invokes . . . .   | csh(C)        |
| getc, getcb, getcbp, getcf: read | clist buffers . . . . .              | getc(K)       |
| putc, putcbp, putcf: write to    | clists putc, . . . . .               | putc(K)       |
| alarm: sets a process' alarm     | clock . . . . .                      | alarm(S)      |
| /the frequency of the system     | clock in ticks per second . . . .    | gethz(S)      |
| rtc: real time                   | clock interface . . . . .            | rtc(HW)       |
|                                  | clock: reports CPU time used . . .   | clock(S)      |
| (time of day) clock              | clock: the system real-time . . . .  | clock(F)      |
| system real-time (time of day)   | clock clock: the . . . . .           | clock(F)      |
| gets string from real-time       | clock getclk: . . . . .              | getclk(M)     |
| system real-time (time of day)   | clock setclock: sets the . . . . .   | setclock(ADM) |
| operations                       | closedir: performs directory . . .   | directory(S)  |
| close:                           | closes a file descriptor . . . . .   | close(S)      |
| files in the current/ rmtmp:     | closes and deletes all temporary .   | rmtmp(DOS)    |
| fclose, fflush:                  | closes or flushes a stream . . . .   | fclose(S)     |
| shuts down the/ haltsys, reboot: | closes out the filesystems and . .   | haltsys(ADM)  |
| fclose, fcloseall:               | closes streams . . . . .             | fclose(DOS)   |
|                                  | clri: clears inode . . . . .         | clri(ADM)     |
| size                             | cmchk: reports hard disk block . .   | cmchk(C)      |
| panics the system                | cmn_err: displays message or . . .   | cmn_err(K)    |
| configuration data base          | cmos: displays and sets the . . . .  | cmos(HW)      |
|                                  | cmp: compares two files . . . . .    | cmp(C)        |
| dis: object                      | code disassembler . . . . .          | dis(CP)       |
| sets u.u_error with error        | code seterror: . . . . .             | seterror(K)   |
|                                  | codeview: visual debugger . . . .    | codeview(CP)  |
| coffconv: convert 386            | COFF files to XENIX format . . . .   | coffconv(M)   |
|                                  | col: filters reverse linefeeds . . . | col(C)        |
| coltbl: create a                 | collation locale table . . . . .     | coltbl(M)     |
| strcoll, strcoll: handles        | collation of strings /strnxfm, . .   | collation(S)  |
| the audit/ auditd: read audit    | collection files generated by . . .  | auditd(ADM)   |
| screen: tty[01-n],               | color, monochrome, ega, . . . .      | screen(HW)    |
| setcolor: set screen             | color . . . . .                      | setcolor(C)   |



|                                  |                                      |               |
|----------------------------------|--------------------------------------|---------------|
| locale table                     | coltbl: create a collation . . . . . | coltbl(M)     |
| lc: lists directory contents in  | columns . . . . .                    | lc(C)         |
| comb:                            | comb: combines SCCS deltas . . .     | comb(CP)      |
| comb: combines SCCS deltas       | comb(CP)                             | comb(CP)      |
| common to two sorted files       | comm: selects or rejects lines . .   | comm(C)       |
| nice: runs a                     | command at a different priority . .  | nice(C)       |
| /putprcmnam: manipulate          | command control database entry . .   | getprcmnt(S)  |
| segread:                         | command description . . . . .        | segread(DOS)  |
| env: sets environment for        | command execution . . . . .          | env(C)        |
| quits nohup: runs a              | command immune to hangups and . .    | nohup(C)      |
| subsystem activation,/ auditcmd: | command interface for audit . . .    | auditcmd(ADM) |
| rsh: invokes a restricted shell  | (command interpreter) . . . . .      | rsh(C)        |
| sh: invokes the shell            | command interpreter . . . . .        | sh(C)         |
| syntax csh: invokes a shell      | command interpreter with C-like . .  | csh(C)        |
| uux: executes                    | command on remote UNIX . . . . .     | uux(C)        |
| getopt: parses                   | command options . . . . .            | getopt(C)     |
| getopts, getoptcv - parse        | command options getopts: . . . .     | getopts(C)    |
| system activity timex: time a    | command; report process data and .   | timex(ADM)    |
| accounting records acctcms:      | command summary from per-process     | acctcms(ADM)  |
| system: executes a shell         | command . . . . .                    | system(S)     |
| time: times a                    | command . . . . .                    | time(CP)      |
| changes root directory for       | command chroot: . . . . .            | chroot(ADM)   |
| create a front-end to the cc     | command gcc: . . . . .               | gcc(CP)       |
| at, batch: executes              | commands at a later time . . . . .   | at(C)         |
| cron: executes                   | commands at specified times . . .    | cron(C)       |
| micnet: the Micnet default       | commands file . . . . .              | micnet(F)     |
| help: asks for help about SCCS   | commands . . . . .                   | help(CP)      |
| intro: introduces UNIX           | commands . . . . .                   | Intro(C)      |
| system remote: executes          | commands on a remote UNIX . . .      | remote(C)     |
| environment rc2: run             | commands performed for multiuser     | rc2(ADM)      |
| operating system rc0: run        | commands performed to stop the .     | rc0(ADM)      |
| xargs: constructs and executes   | commands . . . . .                   | xargs(C)      |
| UNIX Development System          | commands intro: introduces . . .     | Intro(CP)     |
| and miscellaneous accounting     | commands /overview of accounting     | acct(ADM)     |
| interpret tty driver I/O control | commands tticom: . . . . .           | tticom(K)     |
| executes an operating system     | command system: . . . . .            | system(DOS)   |
| mcs: manipulate the object file  | comment section . . . . .            | mcs(CP)       |
| cdc: changes the delta           | commentary of an SCCS delta . .      | cdc(CP)       |
| as:                              | common assembler . . . . .           | as(CP)        |
| convert archive files to         | common formats convert: . . . .      | convert(CP)   |
| routines ldfcn:                  | common object file access . . . .    | ldfcn(F)      |
| conv:                            | common object file converter . . .   | conv(CP)      |
| cprs: compress a                 | common object file . . . . .         | cprs(CP)      |
| ldopen, ldaopen: open a          | common object file for reading . .   | ldopen(S)     |
| /line number entries of a        | common object file function . . .    | ldlread(S)    |
| ldclose, ldclose: close a        | common object file . . . . .         | ldclose(S)    |
| /section header of a             | common object file . . . . .         | ldshread(S)   |
| scnhdr: section header for a     | common object file . . . . .         | scnhdr(F)     |
| entry /retrieve symbol name for  | common object file symbol table .    | ldgetname(S)  |
| format syms:                     | common object file symbol table .    | syms(F)       |
| read the file header of a        | common object file ldhread: . . .    | ldhread(S)    |

|                                  |                                        |               |
|----------------------------------|----------------------------------------|---------------|
| seek to the symbol table of a    | common object file ldtbseek:           | ldtbseek(S)   |
| line number entries in a         | common object file linenum:            | linenum(F)    |
| produce C source listing from a  | common object file list:               | list(CP)      |
| indexed symbol table entry of a  | common object file /read an            | ldtbread(S)   |
| relocation information for a     | common object file reloc:              | reloc(F)      |
| entries of a section of a        | common object file /relocation         | ldrseek(S)    |
| filehdr: file header for         | common object files                    | filehdr(F)    |
| to the optional file header of a | common object file /seek               | ldohseek(S)   |
| an indexed/named section of a    | common object file /seek to            | ldsseek(S)    |
| number entries of a section of a | common object file /seek to line       | ldlseek(S)    |
| of a symbol table entry of a     | common object file /the index          | ldtbindex(S)  |
| comm: selects or rejects lines   | common to two sorted files             | comm(C)       |
| /the status of inter-process     | communication facilities               | ipcs(ADM)     |
| ftok: standard interprocess      | communication package                  | stdipc(S)     |
| nl_strcmp, nl_strncmp:           | compare native language strings        | nl_strcmp(S)  |
| descriptions                     | infocmp: compare or print out terminfo | infocmp(ADM)  |
| chkshlib:                        | compare shared libraries tool          | chkshlib(CP)  |
| memicmp:                         | compares buffers byte-by-byte          | memicmp(DOS)  |
| dircmp:                          | compares directories                   | dircmp(C)     |
| sdiff:                           | compares files side-by-side            | sdiff(C)      |
| diff bdiff:                      | compares files too large for           | bdiff(C)      |
| diskcp, diskcmp: copies or       | compares floppy disks                  | diskcp(C)     |
| diff3:                           | compares three files                   | diff3(C)      |
| cmp:                             | compares two files                     | cmp(C)        |
| diff:                            | compares two text files                | diff(C)       |
| file scsdiff:                    | compares two versions of an SCCS       | scsdiff(CP)   |
| expression regcmp, regex:        | compile and execute regular            | regcmp(S)     |
| regex: regular expression        | compile and match routines             | regex(S)      |
| terminfo: format of              | compiled terminfo file                 | terminfo(F)   |
| cc: invokes the C                | compiler                               | cc(CP)        |
| rcc: AT&T C                      | compiler                               | rcc(CP)       |
| tic: terminfo                    | compiler                               | tic(C)        |
| yacc: invokes a                  | compiler-compiler                      | yacc(CP)      |
| expressions regex, regcmp:       | compiles and executes regular          | regex(S)      |
| regcmp:                          | compiles regular expressions           | regcmp(CP)    |
| erf, erfc: error function and    | complementary error function           | erf(S)        |
| iodone: signals I/O              | completion                             | iodone(K)     |
| iowait: wait for I/O             | completion                             | iowait(K)     |
| processes wait: awaits           | completion of background               | wait(C)       |
| the absolute value of a          | complex number cabs: calculates        | cabs(DOS)     |
| subsystem: security subsystem    | component description                  | subsystem(M)  |
| cprs:                            | compress a common object file          | cprs(CP)      |
| storage                          | compress: compress data for            | compress(C)   |
| compress:                        | compress data for storage              | compress(C)   |
| pack, pcatt, unpack:             | compresses and expands files           | pack(C)       |
| table entry of a/ ldtbindex:     | compute the index of a symbol          | ldtbindex(S)  |
| scsi: small                      | computer systems interface             | scsi(HW)      |
| time values difftime:            | computes the difference between        | difftime(DOS) |
| time values difftime:            | computes the difference between        | difftime(S)   |
| cat:                             | concatenates and displays files        | cat(C)        |
|                                  | conditions. test: tests                | test(C)       |



|                                  |                                  |                   |
|----------------------------------|----------------------------------|-------------------|
| pathconf: get                    | configurable pathname variables  | pathconf(S)       |
| sysconf: get                     | configurable system variables    | sysconf(S)        |
| cmos: displays and sets the      | configuration data base          | cmos(HW)          |
| update, or get device driver     | configuration data /add, delete, | idinstall(ADM)    |
| filesystem types mfsys:          | configuration file for           | mfsys(F)          |
| mvdevice: video driver backend   | configuration file               | mvdevice(F)       |
| sdevice: local device            | configuration file               | sdevice(F)        |
| hwconfig: read the               | configuration information        | hwconfig(ADM)     |
| /mapscrm, mapstr, convkey:       | configure monitor screen/        | mapkey(M)         |
| mapchan:                         | configure tty device mapping     | mapchan(M)        |
| spooling system lpadmin:         | configures the lineprinter       | lpadmin(ADM)      |
| t_rcvconnect: receive the        | confirmation from a connect/     | t_rcvconnect(NSL) |
| tracing a/ paccess: used in      | conjunction with ptrace for      | paccess(S)        |
| /fwtmp, wtmpfix: manipulate      | connect accounting records       | fwtmp(ADM)        |
| t_accept: accept a               | connect request                  | t_accept(NSL)     |
| t_listen: listen for a           | connect request                  | t_listen(NSL)     |
| receive the confirmation from a  | connect request t_rcvconnect:    | t_rcvconnect(NSL) |
| t_connect: establish a           | connection with another/         | t_connect(NSL)    |
| an out-going terminal line       | connection dial: establishes     | dial(S)           |
| or expedited data sent over a    | connection t_rcv: receive data   | t_rev(NSL)        |
| data or expedited data over a    | connection t_snd: send           | t_snd(NSL)        |
| /_nheapchk: performs a minimal   | consistency check on the heap    | _fheapchk(DOS)    |
| database authck: check internal  | consistency of Authentication    | authck(ADM)       |
| returns a character to the       | console buffer ungetch:          | ungetch(DOS)      |
| cputs: puts a string to the      | console                          | cputs(DOS)        |
| console: system                  | console device                   | console(M)        |
| kbhit: checks the                | console for a keystroke          | kbhit(DOS)        |
| cscanf: converts and formats     | console input                    | cscanf(DOS)       |
| messages: description of system  | console messages                 | messages(M)       |
| printf: print a message on the   | console                          | printf(K)         |
| putch: writes a character to the | console                          | putch(DOS)        |
|                                  | console: system console device   | console(M)        |
| a device error message on the    | console devert: prints           | devert(K)         |
| to printer attached to a serial  | console /Print file              | consoleprint(ADM) |
| printer attached to a serial/    | consoleprint: print file to      | consoleprint(ADM) |
| prints a character on the        | console putchar:                 | putchar(K)        |
| math: math functions and         | constants                        | math(M)           |
| unistd: file header for symbolic | constants                        | unistd(F)         |
| for implementation-specific      | constants limits: file header    | limits(F)         |
| mkfs:                            | constructs a filesystem          | mkfs(ADM)         |
| commands xargs:                  | constructs and executes          | xargs(C)          |
| debugging on uutry: try to       | contact remote system with       | uutry(ADM)        |
| idmkinit: read files             | containing specifications        | idmkinit(ADM)     |
| ev_block: wait until the queue   | contains an event                | ev_block(S)       |
| lc: lists directory              | contents in columns              | lc(C)             |
| ls: gives information about      | contents of directories          | ls(C)             |
| l: lists information about       | contents of directory            | l(C)              |
| splits files according to        | context csplit:                  | csplit(C)         |
| memget: allocates                | contiguous memory at/            | memget(K)         |
| allocates and frees physically   | contiguous memory /db_free:      | db_alloc(K)       |
| transfers from a user address to | contiguous memory db_write:      | db_write(K)       |



## Permuted Index

|                                  |                                          |                   |                |
|----------------------------------|------------------------------------------|-------------------|----------------|
| interpret tty driver I/O         | control commands                         | ttiocom: . . .    | ttiocom(K)     |
| /putprcmnam: manipulate command  | control database entry . . . . .         |                   | getprcmnt(S)   |
| /putprdfnam: manipulate default  | control database entry . . . . .         |                   | getprdfent(S)  |
| /putprfinam: manipulate file     | control database entry . . . . .         |                   | getprfient(S)  |
| /putprtcnam: manipulate terminal | control database entry . . . . .         |                   | getprtcnt(S)   |
| UUCP                             | control files. uuinstall: administrators |                   | uuinstall(ADM) |
| device tapecntl: AT&T tape       | control for QIC-24/QIC-02 tape . .       | tapecntl(C)       |                |
| tcflush, tcsendbreak: line       | control functions /tcflow, . . . .       | tcflow(S)         |                |
| init, inir: process              | control initialization . . . . .         | init(M)           |                |
| jagent: host                     | control of windowing terminal . .        | jagent(M)         |                |
| msgctl: provides message         | control operations . . . . .             | msgctl(S)         |                |
| fcntl: file                      | control options . . . . .                | fcntl(M)          |                |
| and size setvbuf: allow user     | control over stream buffering . . .      | setvbuf(DOS)      |                |
| uadmin: administrative           | control . . . . .                        | uadmin(ADM)       |                |
| uadmin: administrative           | control . . . . .                        | uadmin(S)         |                |
| vc: version                      | control . . . . .                        | vc(C)             |                |
| vc: version                      | control . . . . .                        | vc(CP)            |                |
| gets and sets floating-point     | control word _control87: . . . . .       | _control87(DOS)   |                |
| floating-point control word      | _control87: gets and sets . . . .        | _control87(DOS)   |                |
| IEEE floating point environment  | control /fpsetsticky: . . . . .          | fpgetround(S)     |                |
| dma_param: sets up a DMA         | controller chip for DMA transfer .       | dma_param(K)      |                |
| ioctl:                           | controls character devices . . . .       | ioctl(S)          |                |
| fcntl:                           | controls open files . . . . .            | fcntl(S)          |                |
| semctl:                          | controls semaphore operations . .        | semctl(S)         |                |
| operations shmctl:               | controls shared memory . . . . .         | shmctl(S)         |                |
| set process group ID for job     | control setpgid: . . . . .               | setpgid(S)        |                |
| uucp status inquiry and job      | control uustat: . . . . .                | uustat(C)         |                |
| converter                        | conv: common object file . . . . .       | conv(CP)          |                |
| translates characters            | conv, toupper, tolower, toascii: .       | conv(S)           |                |
| binary/ /fmsbintoieee: perform   | conversions between IEEE and MS          | fiieetomsbin(DOS) |                |
| and IEEE/ /dmsbintoieee: perform | conversions between MS binary .          | dieeetomsbin(DOS) |                |
| fcvt, gcv: performs output       | conversions ecvt, . . . . .              | ecvt(S)           |                |
| format coffconv:                 | convert 386 COFF files to XENIX .        | coffconv(M)       |                |
| into a terminfo/ captinfo:       | convert a termcap description . .        | captinfo(ADM)     |                |
| physical address vtop:           | convert a virtual address to a . .       | vtop(K)           |                |
| formats convert:                 | convert archive files to common .        | convert(CP)       |                |
| to common formats                | convert: convert archive files . .       | convert(CP)       |                |
| conv: common object file         | converter . . . . .                      | conv(CP)          |                |
| double-precision/ strtod, atof:  | converts a string to a . . . . .         | strtod(S)         |                |
| routing file/ mnlst:             | converts a XENIX-style Micnet . .        | mnlst(ADM)        |                |
| UUCP routing/ uulst:             | converts a XENIX-style . . . . .         | uulst(ADM)        |                |
| dd:                              | converts and copies a file . . . .       | dd(C)             |                |
| input cscanf:                    | converts and formats console . . .       | csanf(DOS)        |                |
| scanf, fscanf, sscanf:           | converts and formats input . . . .       | scanf(S)          |                |
| libraries ranlib:                | converts archives to random . . .        | ranlib(CP)        |                |
| atof, atoi, atol:                | converts ASCII to numbers . . . .        | atof(S)           |                |
| and long/ l3tol, ltol3:          | converts between 3-byte integers .       | l3tol(S)          |                |
| clicks/ btoc, btoms, ctob:       | converts between bytes and . . . .       | btoc(K)           |                |
| and base 64 ASCII a64l, l64a:    | converts between long integer . .        | a64l(S)           |                |
| toupper, toascii: classifies or  | converts characters /tolower, . .        | ctype(S)          |                |
| /gmtime, asctime, tzset:         | converts date and time to ASCII .        | ctime(S)          |                |

|                                  |                                       |                |
|----------------------------------|---------------------------------------|----------------|
| nl_scanf, nl_fscanf, nl_sscanf:  | converts formatted native/ . . . .    | nl_scanf(S)    |
| time mktime:                     | converts local time to calendar . . . | mktime(S)      |
| characters ltoa:                 | converts long integers to . . . .     | ltoa(DOS)      |
| uppercase strupr:                | converts lowercase characters to . .  | strupr(DOS)    |
| ultoa:                           | converts numbers to characters . . .  | ultoa(DOS)     |
| itoa:                            | converts numbers to integers . . . .  | itoa(DOS)      |
| standard FORTRAN ratfor:         | converts Rational FORTRAN into . .    | ratfor(CP)     |
| strtol, atol, atoi:              | converts string to integer . . . .    | strtol(S)      |
| labs:                            | converts to absolute value . . . .    | labs(S)        |
| units:                           | converts units . . . . .              | units(C)       |
| lowercase strlwr:                | converts uppercase characters to . .  | strlwr(DOS)    |
| addresses ptok, ktcp:            | converts virtual and physical . . .   | ptok(K)        |
| file to MMDF/ mmdfalias:         | converts XENIX-style aliases . . .    | mmdfalias(ADM) |
| screen/ mapkey, mapscrn, mapstr, | convkey: configure monitor . . . .    | mapkey(M)      |
| dd: converts and                 | copies a file . . . . .               | dd(C)          |
| kernel space copyin, copyout:    | copies bytes between user and . . .   | copyin(K)      |
| address movedata:                | copies bytes from a specific . . .    | movedata(DOS)  |
| bcopy:                           | copies bytes in kernel space . . . .  | bcopy(K)       |
| physical address copyio:         | copies bytes to and from a . . . .    | copyio(K)      |
| objects memmove:                 | copies characters between . . . .     | memmove(DOS)   |
| objects memmove:                 | copies characters between . . . .     | memmove(S)     |
| cpio:                            | copies file archives in and out . .   | cpio(C)        |
| systems rcp:                     | copies files across UNIX . . . .      | rcp(C)         |
| cp:                              | copies files . . . . .                | cp(C)          |
| copy:                            | copies groups of files . . . . .      | copy(C)        |
| diskcp, diskcmp:                 | copies or compares floppy disks . .   | diskcp(C)      |
|                                  | copy: copies groups of files . . . .  | copy(C)        |
| volcopy: make literal            | copy of UNIX filesystem . . . . .     | volcopy(ADM)   |
| for optimal access time dcopy:   | copy UNIX filesystems . . . . .       | dcopy(ADM)     |
| /enddvagent, putdvagnam,         | copydvagent: manipulate device/ . .   | getdvagent(S)  |
| between user and kernel space    | copyin, copyout: copies bytes . . .   | copyin(K)      |
| a physical address               | copyio: copies bytes to and from . .  | copyio(K)      |
| user and kernel space copyin,    | copyout: copies bytes between . . .   | copyin(K)      |
| public UNIX-to-UNIX file         | copy uuto, uupick: . . . . .          | uuto(C)        |
|                                  | core: format of core image file . . . | core(F)        |
| core: format of                  | core image file . . . . .             | core(F)        |
| asktime: prompts for the         | correct time of day . . . . .         | asktime(ADM)   |
| atan2: performs/ sin,            | cos, tan, asin, acos, atan, . . . .   | trig(S)        |
| functions sinh,                  | cosh, tanh: performs hyperbolic . .   | sinh(S)        |
| display line-by-line execution   | count profile data lprof: . . . . .   | lprof(CP)      |
| _freect:                         | counts available dynamic memory . .   | _freect(DOS)   |
| sum: calculates checksum and     | counts blocks in a file . . . . .     | sum(C)         |
| characters wc:                   | counts lines, words and . . . . .     | wc(C)          |
|                                  | cp: copies files . . . . .            | cp(C)          |
| between user space and the/      | cpass, passc: passes character . . .  | cpass(K)       |
| cpio: format of                  | cpio archive . . . . .                | cpio(F)        |
| and out                          | cpio: copies file archives in . . . . | cpio(C)        |
|                                  | cpio: format of cpio archive . . . .  | cpio(F)        |
| preprocessor                     | cpr: the C language . . . . .         | cpr(CP)        |
|                                  | cprintf: formats output . . . . .     | cprintf(DOS)   |
| file                             | cprs: compress a common object . .    | cprs(CP)       |



|                                 |                                        |               |
|---------------------------------|----------------------------------------|---------------|
| clock: reports                  | cPU time used . . . . .                | clock(S)      |
| flushes block I/O and halts the | cPU shutdn: . . . . .                  | shutdown(S)   |
| console                         | cputs: puts a string to the . . . .    | cputs(DOS)    |
|                                 | crash: examine system images . . .     | crash(ADM)    |
| overwrites an existing one      | creat: creates a new file or . . . .   | creat(DOS)    |
| rewrites an existing one        | creat: creates a new file or . . . .   | creat(S)      |
| coltbl:                         | create a collation locale table . . .  | coltbl(M)     |
| chrtbl:                         | create a ctype locale table . . . .    | chrtbl(M)     |
| montbl:                         | create a currency locale table . . .   | montbl(M)     |
| command gcc:                    | create a front-end to the cc . . . .   | gcc(CP)       |
| mestbl:                         | create a messages locale file . . . .  | mestbl(M)     |
| numtbl:                         | create a numeric locale table . . . .  | numtbl(M)     |
| mkshlib:                        | create a shared library . . . . .      | mkshlib(CP)   |
| ID setuid:                      | create session and set process . . .   | setuid(S)     |
| file tmpnam, tempnam:           | creates a name for a temporary . . .   | tmpnam(S)     |
| mkdir:                          | creates a new directory . . . . .      | mkdir(DOS)    |
| an existing one creat:          | creates a new file or overwrites . .   | creat(DOS)    |
| an existing one creat:          | creates a new file or rewrites . . .   | creat(S)      |
| fork:                           | creates a new process . . . . .        | fork(S)       |
| spawnl, spawnvp:                | creates a new process . . . . .        | spawn(DOS)    |
| ctags:                          | creates a tags file . . . . .          | ctags(CP)     |
| tee:                            | creates a tee in a pipe . . . . .      | tee(C)        |
| tmpfile:                        | creates a temporary file . . . . .     | tmpfile(S)    |
| from C source mkstr:            | creates an error message file . . . .  | mkstr(CP)     |
| profile profil:                 | creates an execution time . . . . .    | profil(S)     |
| semaphore creatsem:             | creates an instance of a binary . . .  | creatsem(S)   |
| pipe:                           | creates an interprocess pipe . . . .   | pipe(S)       |
| files admin:                    | creates and administers SCCS . . . .   | admin(CP)     |
| /Scans fixed disk for flaws and | creates bad track table . . . . .      | badtrk(ADM)   |
| umask: sets and gets file       | creation mask . . . . .                | umask(S)      |
| a binary semaphore              | creatsem: creates an instance of . . . | creatsem(S)   |
| listing                         | cref: makes a cross-reference . . . .  | cref(CP)      |
| atcronsh: menu driven at and    | cron administration utility . . . . .  | atcronsh(ADM) |
| specified times                 | cron: executes commands at . . . .     | cron(C)       |
| crontab: user                   | crontab file . . . . .                 | crontab(C)    |
|                                 | crontab: user crontab file . . . . .   | crontab(C)    |
| intro: introduction to DOS      | cross development functions . . . .    | intro(DOS)    |
| dosld: XENIX to MS-DOS          | cross linker . . . . .                 | dosld(CP)     |
| os2ld: OS/2                     | cross linker . . . . .                 | os2ld(CP)     |
| cxref: generates C program      | cross-reference . . . . .              | cxref(CP)     |
| cref: makes a                   | cross-reference listing . . . . .      | cref(CP)      |
| xref:                           | cross-references C programs . . . .    | xref(CP)      |
| item:                           | CRT item routines . . . . .            | item(S)       |
| menu:                           | CRT menu routines . . . . .            | menu(S)       |
|                                 | crypt: encode/decode . . . . .         | crypt(C)      |
| encryption functions            | crypt: password and file . . . . .     | crypt(S)      |
| determine if password is        | cryptic acceptable_password: . . . .   | accept_pw(S)  |
| console input                   | cscanf: converts and formats . . . .   | cscanf(DOS)   |
| C program                       | cscope: interactively examine a . . .  | cscope(CP)    |
| interpreter with C-like syntax  | csh: invokes a shell command . . . .   | csh(C)        |
| to context                      | csplit: splits files according . . . . | csplit(C)     |



|                                   |                                       |                    |
|-----------------------------------|---------------------------------------|--------------------|
| terminal                          | ct: spawn getty to a remote . . .     | ct(C)              |
|                                   | ctags: creates a tags file . . .      | ctags(CP)          |
| for a terminal                    | ctermid: generates a filename . . .   | ctermid(S)         |
| asctime, tzset: converts date/    | ctime, localtime, gmtime, . . .       | ctime(S)           |
| clicks (memory/ btoc, btoms,      | ctob: converts between bytes and .    | btoc(K)            |
|                                   | ctrace: C program debugger . . .      | ctrace(CP)         |
| islower, isdigit, isxdigit,/      | ctype, isalpha, isupper, . . .        | ctype(S)           |
| chrtbl: create a                  | ctype locale table . . .              | chrtbl(M)          |
|                                   | cu: calls another UNIX system . .     | cu(C)              |
| montbl: create a                  | currency locale table . . .           | montbl(M)          |
| all temporary files in the        | current directory /and deletes . .    | rmtmp(DOS)         |
| ev_getemask: return the           | current event mask . . .              | ev_getemask(S)     |
| ev_getemask: return the           | current event mask . . .              | ev_gtemask(S)      |
| endpoint t_look: look at the      | current event on a transport . . .    | t_look(NSL)        |
| rename login entry to show        | current layer relogin: . . .          | relogin(ADM)       |
| pointer fiell: finds the          | current position of a file . . .      | ftell(DOS)         |
| pointer tell: gets the            | current position of the file . . .    | tell(DOS)          |
| activity sact: prints             | current SCCS file editing . . .       | sact(CP)           |
| t_getstate: get the               | current state . . .                   | t_getstate(NSL)    |
| longjmp: ends                     | current system call with error . .    | longjmp(K)         |
| fime: gets the                    | current time . . .                    | fime(DOS)          |
| uname: prints the name of the     | current UNIX system . . .             | uname(C)           |
| uname: gets name of               | current UNIX system . . .             | uname(S)           |
| suser: determines if              | current user is the super-user . .    | suser(K)           |
| the slot in the utmp file of the  | current user tty slot: finds . . .    | ttyslot(S)         |
| fgetpos: gets and stores the      | current value of a stream's file/ . . | fgetpos(DOS)       |
| fgetpos: gets and stores the      | current value of a stream's file/ . . | fgetpos(S)         |
| chdir: changes                    | current working directory . . .       | chdir(DOS)         |
| getcwd: get the pathname of       | current working directory . . .       | getcwd(S)          |
| /Returns the number of events     | currently in the queue . . .          | ev_count(S)        |
| ev_flush: discard all events      | currently in the queue . . .          | ev_flush(S)        |
| /the list of major device numbers | currently specified in the/ . . .     | major sinuse(ADM)  |
| /displays the list of vectors     | currently specified in the/ . . .     | vector sinuse(ADM) |
| cursor functions                  | curses: performs screen and . . .     | curses(S)          |
| scr_dump: format of               | curses screen image file . . .        | scr_dump(F)        |
| curses: performs screen and       | cursor functions . . .                | curses(S)          |
| spline: interpolates smooth       | curve . . .                           | spline(C)          |
| the user                          | cuserid: gets the login name of . .   | cuserid(S)         |
| each line of a file               | cut: cuts out selected fields of . .  | cut(C)             |
| line of a file cut:               | cuts out selected fields of each . .  | cut(C)             |
| cross-reference                   | cxref: generates C program . . .      | cxref(CP)          |
| daemon.mn: micnet mailer          | daemon . . .                          | daemon.mn(M)       |
|                                   | daemon.mn: micnet mailer daemon .     | daemon.mn(M)       |
| STREAMS error logger              | daemon strerr: . . .                  | strerr(ADM)        |
| runacct: run                      | daily accounting . . .                | runacct(ADM)       |
| - handle special functions of     | DASI 300 and 300s/ /300s . . .        | 300(C)             |
| handle special functions of the   | DASI 450 terminal 450: . . .          | 450(C)             |
| sdwaitv: synchronizes shared      | data access sdgetv, . . .             | sdgetv(S)          |
| reduce: perform audit             | data analysis and reduction . . .     | reduce(ADM)        |
| time a command; report process    | data and system activity timex: . .   | timex(ADM)         |
| termcap: terminal capability      | data base . . .                       | termcap(M)         |

|                                  |                                      |                 |
|----------------------------------|--------------------------------------|-----------------|
| terminfo: terminal capability    | data base . . . . .                  | terminfo(M)     |
| and sets the configuration       | data base cmos: displays . . . .     | cmos(HW)        |
| generate disk accounting         | data by user ID diskusg: . . . .     | diskusg(ADM)    |
| t_rcvuderr: receive a unit       | data error indication . . . . .      | t_rcvuderr(NSL) |
| compress: compress               | data for storage . . . . .           | compress(C)     |
| user address db_read: transfers  | data from physical memory to a . .   | db_read(K)      |
| canon: processes raw input       | data from tty device . . . . .       | canon(K)        |
| brkctl: allocates                | data in a far segment . . . . .      | brkctl(S)       |
| /sgetl: accesses long integer    | data in a machine-independent . .    | sputl(S)        |
| plock: lock process, text, or    | data in memory . . . . .             | plock(S)        |
| /open and access audit session   | data on a record basis . . . . .     | audit(S)        |
| connection t_snd: send           | data or expedited data over a . . .  | t_snd(NSL)      |
| a connection t_rcv: receive      | data or expedited data sent over . . | t_rcv(NSL)      |
| t_snd: send data or expedited    | data over a connection . . . . .     | t_snd(NSL)      |
| prof: displays profile           | data . . . . .                       | prof(CP)        |
| prof: displays profile           | data . . . . .                       | prof(XNX)       |
| execseg: makes a                 | data region executable . . . . .     | execseg(S)      |
| call stat:                       | data returned by stat system . . .   | stat(F)         |
| brk, sbrk: change                | data segment space allocation . .    | brk(S)          |
| sbrk, brk: changes               | data segment space allocation . .    | sbrk(S)         |
| synchronizes access to a shared  | data segment sdenter, sdleave: . .   | sdenter(S)      |
| attaches and detaches a shared   | data segment sdget, sdfree: . . .    | sdget(S)        |
| t_rcv: receive data or expedited | data sent over a connection . . .    | t_rcv(NSL)      |
| gets a character from user       | data space fubyte: . . . . .         | fubyte(K)       |
| gets one 32-bit word from user   | data space fuword: . . . . .         | fuword(K)       |
| stores a character in user       | data space subyte: . . . . .         | subyte(K)       |
| stores a 32-bit word in user     | data space suword: . . . . .         | suword(K)       |
| rdchk: checks to see if there is | data to be read . . . . .            | rdchk(S)        |
| types: primitive system          | data types . . . . .                 | types(F)        |
| t_rcvudata: receive a            | data unit . . . . .                  | t_rcvudata(NSL) |
| t_sndudata: send a               | data unit . . . . .                  | t_sndudata(NSL) |
| get device driver configuration  | data /add, delete, update, or . . .  | idinstall(ADM)  |
| authcap: authentication          | database . . . . .                   | authcap(ADM)    |
| manipulate device assignment     | database entry /copydvagent: . .     | getdvagent(S)   |
| manipulate command control       | database entry /putprcmnam: . . .    | getprcmnt(S)    |
| manipulate default control       | database entry /putprdfnam: . . .    | getprdfent(S)   |
| manipulate file control          | database entry /putprfinam: . . .    | getprfient(S)   |
| manipulate protected password    | database entry /putprpwnam: . . .    | getprpwent(S)   |
| manipulate terminal control      | database entry /putprtcnam: . . .    | getprtcent(S)   |
| backups schedule:                | database for automated system . .    | schedule(ADM)   |
| firstkey, nextkey: performs      | database functions /delete, . . .    | dbm(S)          |
| tables nictable: process NIC     | database into channel/domain . .     | nictable(ADM)   |
| routines for Subsystems          | database /: manipulation . . . .     | subsystems(S)   |
| /builds the MMDF hashed          | database of alias and routing/ . .   | dbmbuild(ADM)   |
| terminfo: terminal description   | database . . . . .                   | terminfo(S)     |
| terminfo: terminal description   | database . . . . .                   | terminfo(S)     |
| tput: queries the terminfo       | database . . . . .                   | tput(C)         |
| consistency of Authentication    | database authck: check internal .    | authck(ADM)     |
| lock the entire Authentication   | database dblock: . . . . .           | dblock(S)       |
| files against authentication     | database /examine system . . . .     | integrity(ADM)  |
| from the authentication          | database /get information . . . .    | authcap(S)      |



|                                   |                                      |                |
|-----------------------------------|--------------------------------------|----------------|
| on fields of authentication       | database /return status based . . .  | fields(S)      |
| execution count profile           | data /display line-by-line . . .     | lprof(CP)      |
| virtual address pointer to block  | data paddr: returns . . .            | paddr(K)       |
| and writes streams of device      | data /repoutsw, repoutsd: reads . .  | repins(K)      |
| nl_ascxtime, nl_cxtime: format    | date and time . . .                  | nl_cxtime(S)   |
| /gmtime, asctime, tzset: converts | date and time to ASCII . . .         | ctime(S)       |
| date: prints and sets the         | date . . .                           | date(C)        |
|                                   | date: prints and sets the date . .   | date(C)        |
| time, ftime: gets time and        | date . . .                           | time(S)        |
| the access and modification       | dates of files /Changes . . .        | settime(ADM)   |
| sddate: prints and sets backup    | dates . . .                          | sddate(C)      |
|                                   | date/time string . . .               | strftime(S)    |
| the system real-time (time of     | day) clock clock: . . .              | clock(F)       |
| the system real-time (time of     | day) clock setclock: sets . . .      | setclock(ADM)  |
| prompts for the correct time of   | day asktime: . . .                   | asktime(ADM)   |
| freed physically contiguous/      | db_alloc, db_free: allocates and . . | db_alloc(K)    |
| physically contiguous/ db_alloc,  | db_free: allocates and frees . . .   | db_alloc(K)    |
| Authentication database           | dblock: lock the entire . . .        | dblock(S)      |
| MMDF hashed database of/          | dbmbuild: builds the . . .           | dbmbuild(ADM)  |
| firstkey, nextkey: performs/      | dbmunit, fetch, store, delete, . .   | dbm(S)         |
| physical memory to a user/        | db_read: transfers data from . . .   | db_read(K)     |
| address to contiguous memory      | db_write: transfers from a user . .  | db_write(K)    |
| precision calculator              | dc: invokes an arbitrary . . .       | dc(C)          |
| filesystems for optimal access/   | dcopy: copy UNIX . . .               | dcopy(ADM)     |
|                                   | dd: converts and copies a file . .   | dd(C)          |
| free, _ffree, _nfree:             | deallocate memory blocks . . .       | free(DOS)      |
| hfree:                            | deallocates a memory block . . .     | hfree(DOS)     |
| devices assign,                   | deassign: assigns and deassigns . .  | assign(C)      |
| assign, deassign: assigns and     | deassigns devices . . .              | assign(C)      |
| adb: invokes a general-purpose    | debugger . . .                       | adb(CP)        |
| codeview: visual                  | debugger . . .                       | codeview(CP)   |
| ctrace: C program                 | debugger . . .                       | ctrace(CP)     |
| fsdb: file system                 | debugger . . .                       | fsdb(ADM)      |
| sdb: invokes symbolic             | debugger . . .                       | sdb(CP)        |
| to contact remote system with     | debugging on uutry: try . . .        | uutry(ADM)     |
| transmission via mail uudecode:   | decode a binary file for . . .       | uencode(C)     |
| fdswap: swaps                     | default boot floppy drive . . .      | fdswap(ADM)    |
| micnet: the Micnet                | default commands file . . .          | micnet(F)      |
| /putprdfnam: manipulate           | default control database entry . .   | getprdfent(S)  |
| information directory             | default: default program . . .       | default(F)     |
| defopen, defread: reads           | default entries . . .                | defopen(S)     |
| directory default:                | default program information . . .    | default(F)     |
| timezone: set                     | default system time zone . . .       | timezone(F)    |
| upon receipt of a/ signal:        | defines the action to be taken . .   | signal(DOS)    |
| entries                           | defopen, defread: reads default . .  | defopen(S)     |
| defopen,                          | defread: reads default entries . .   | defopen(S)     |
| for specified time                | delay: delays process execution . .  | delay(K)       |
| specified time delay:             | delays process execution for . . .   | delay(K)       |
| performs/ dbmunit, fetch, store,  | delete, firstkey, nextkey: . . .     | dbm(S)         |
| driver/ idinstall: add,           | delete, update, or get device . . .  | idinstall(ADM) |
| rmdir:                            | deletes a directory . . .            | rmdir(DOS)     |



|                                  |                           |                                                                      |                |
|----------------------------------|---------------------------|----------------------------------------------------------------------|----------------|
|                                  | remove:                   | deletes a file . . . . .                                             | remove(DOS)    |
|                                  | unlink:                   | deletes a file . . . . .                                             | unlink(DOS)    |
| the current/                     | rmtmp:                    | closes and . . . . .                                                 | rmtmp(DOS)     |
|                                  | process                   | deliver: MMDF mail delivery . . . . .                                | deliver(ADM)   |
| which has been submitted but not | delivered                 | /checks for mail . . . . .                                           | checkmail(C)   |
| pathname                         | dirname:                  | delivers directory part of . . . . .                                 | dirname(C)     |
|                                  | file tail:                | delivers the last part of a . . . . .                                | tail(C)        |
|                                  | deliver: MMDF mail        | delivery process . . . . .                                           | deliver(ADM)   |
|                                  | delta:                    | makes a delta (change) to an SCCS file . . . . .                     | delta(CP)      |
| delta                            | cdc:                      | changes the delta commentary of an SCCS . . . . .                    | cdc(CP)        |
|                                  | rm del:                   | removes a delta from an SCCS file . . . . .                          | rm del(CP)     |
|                                  |                           | an SCCS file delta: makes a delta (change) to . . . . .              | delta(CP)      |
| the delta commentary of an SCCS  | delta                     | cdc: changes . . . . .                                               | cdc(CP)        |
|                                  | comb:                     | combines SCCS deltas . . . . .                                       | comb(CP)       |
| terminal                         | mesg:                     | permits or denies messages sent to a . . . . .                       | mesg(C)        |
|                                  | terminfo:                 | terminal description database . . . . .                              | terminfo(S)    |
|                                  | terminfo:                 | terminal description database . . . . .                              | terminfo(S)    |
| captainfo:                       | convert                   | a termcap description into a terminfo/ . . . . .                     | captainfo(ADM) |
|                                  | machine:                  | description of host machine . . . . .                                | machine(HW)    |
|                                  | messages                  | messages: description of system console . . . . .                    | messages(M)    |
|                                  | segread:                  | command description . . . . .                                        | segread(DOS)   |
|                                  |                           | description into a terminfo description /convert a termcap . . . . . | captainfo(ADM) |
| compare or print out terminfo    | descriptions              | infocmp: . . . . .                                                   | infocmp(ADM)   |
| security subsystem component     | description               | subsystem: . . . . .                                                 | subsystem(M)   |
|                                  | close:                    | closes a file descriptor . . . . .                                   | close(S)       |
| dup2:                            | duplicate                 | an open file descriptor . . . . .                                    | dup2(S)        |
| dup2:                            | duplicates                | an open file descriptor dup, . . . . .                               | dup(S)         |
| sdget, sdfree:                   | attaches and              | detaches a shared data segment . . . . .                             | sdget(S)       |
|                                  | fstyp:                    | determine filesystem identifier . . . . .                            | fstyp(ADM)     |
| acceptable_password:             | determine                 | if password is cryptic . . . . .                                     | accept_pw(S)   |
| length                           | passlen:                  | determine minimum password . . . . .                                 | passlen(S)     |
| file                             | access:                   | determines accessibility of a . . . . .                              | access(S)      |
|                                  | dtype:                    | determines disk type . . . . .                                       | dtype(C)       |
|                                  | eof:                      | determines end-of-file . . . . .                                     | eof(DOS)       |
|                                  | hypot, cabs:              | determines Euclidean distance . . . . .                              | hypot(S)       |
|                                  | file:                     | determines file type . . . . .                                       | file(C)        |
| the super-user                   | suser:                    | determines if current user is . . . . .                              | suser(K)       |
| error, feof, clearerr, fileno:   | determines                | stream status . . . . .                                              | ferror(S)      |
| and is accessible                | access:                   | determines whether a file exists . . . . .                           | access(DOS)    |
|                                  | whodo:                    | determines who is doing what . . . . .                               | whodo(C)       |
| message on the console           | deverr:                   | prints a device error . . . . .                                      | deverr(K)      |
| /copydvagent:                    | manipulate                | device assignment database entry . . . . .                           | getdvagent(S)  |
| audit:                           | audit subsystem interface | device . . . . .                                                     | audit(ADM)     |
|                                  | sdevice:                  | local device configuration file . . . . .                            | sdevice(F)     |
| console:                         | system console            | device . . . . .                                                     | console(M)     |
| reads and writes streams of      | device data               | /repoutsw, repoutsd: . . . . .                                       | repins(K)      |
| /add, delete, update, or get     | device driver             | configuration data . . . . .                                         | idinstall(ADM) |
| console                          | deverr:                   | prints a device error message on the . . . . .                       | deverr(K)      |
| error: kernel error output       | device                    | . . . . .                                                            | error(M)       |
| /Default backup                  | device                    | information . . . . .                                                | archive(F)     |
| lp, lp0, lp1, lp2:               | line printer              | device interfaces . . . . .                                          | lp(HW)         |

|                                 |                                       |                   |
|---------------------------------|---------------------------------------|-------------------|
| temporary memory or maps a      | device into memory /allocates . . .   | sptalloc(K)       |
| isatty: checks for a character  | device . . . . .                      | isatty(DOS)       |
| mapchan: format of tty          | device mapping files . . . . .        | mapchan(F)        |
| mapchan: configure tty          | device mapping . . . . .              | mapchan(M)        |
| devnm: identifies               | device name . . . . .                 | devnm(C)          |
| /minor: returns major, new      | device number, or minor device/       | major(K)          |
| device number, or minor         | device number /returns major, . .     | major(K)          |
| /displays the list of major     | device numbers currently/ . . . .     | majorinuse(ADM)   |
| systty: system maintenance      | device . . . . .                      | systty(M)         |
| raw input data from tty         | device canon: processes . . . .       | canon(K)          |
| ev_getdev: gets a list of       | devices feeding an event queue . .    | ev_getdev(S)      |
| devices: format of UUCP         | devices file . . . . .                | devices(F)        |
| ev_gindev: include/exclude      | devices for event input . . . . .     | ev_gindev(S)      |
| file                            | devices: format of UUCP devices .     | devices(F)        |
| ioctl: controls character       | devices . . . . .                     | ioctl(S)          |
| adds a block I/O request to a   | device's queue disksort: . . . .      | disksort(K)       |
| deassign: assigns and deassigns | devices assign, . . . . .             | assign(C)         |
| event queue and all associated  | devices ev_close: close the . . .     | ev_close(S)       |
| control for QIC-24/QIC-02 tape  | device tapectl: AT&T tape . . .       | tapectl(C)        |
| font and video mode for a video | device vidi: sets the . . . . .       | vidi(C)           |
|                                 | devnm: identifies device name . .     | devnm(C)          |
| blocks                          | df: report number of free disk . .    | df(C)             |
|                                 | dial: dials a modem . . . . .         | dial(ADM)         |
| terminal line connection        | dial: establishes an out-going . .    | dial(S)           |
| dialcodes: format of UUCP       | dial-code abbreviations file . . .    | dialcodes(F)      |
| dial-code abbreviations file    | dialcodes: format of UUCP . . .       | dialcodes(F)      |
| dialers: format of UUCP         | dialers file . . . . .                | dialers(F)        |
| file                            | dialers: format of UUCP Dialers .     | dialers(F)        |
| dial:                           | dials a modem . . . . .               | dial(ADM)         |
| uchat:                          | dials a modem . . . . .               | dial(ADM)         |
| passwd: change login, group, or | dialup shell password . . . . .       | passwd(C)         |
| perform conversions between MS/ | diecetomsbin, dmsbintoieee: . . .     | diecetomsbin(DOS) |
|                                 | diff: compares two text files . . .   | diff(C)           |
|                                 | diff3: compares three files . . .     | diff3(C)          |
| compares files too large for    | diff bdiff: . . . . .                 | bdiff(C)          |
| difftime: computes the          | difference between time values . .    | difftime(DOS)     |
| difftime: computes the          | difference between time values . .    | difftime(S)       |
| difference between time values  | difftime: computes the . . . . .      | difftime(DOS)     |
| difference between time values  | difftime: computes the . . . . .      | difftime(S)       |
|                                 | dir: format of a directory . . . .    | dir(F)            |
|                                 | dircmp: compares directories . . .    | dircmp(C)         |
| uuchek: check the uucp          | directories and permissions file . .  | uuchek(ADM)       |
| dircmp: compares                | directories . . . . .                 | dircmp(C)         |
| mv: moves or renames files and  | directories . . . . .                 | mv(C)             |
| rm, rmdir: removes files or     | directories . . . . .                 | rm(C)             |
| rmdir: removes                  | directories . . . . .                 | rmdir(C)          |
| remove temporary files in       | directories specified cleantmp: . . . | cleantmp(ADM)     |
| link and unlink files and       | directories link: link, unlink: . . . | link(ADM)         |
| information about contents of   | directories ls: gives . . . . .       | ls(C)             |
| cd: changes working             | directory . . . . .                   | cd(C)             |
| chdir: changes current working  | directory . . . . .                   | chdir(DOS)        |



|                                 |                                            |                |
|---------------------------------|--------------------------------------------|----------------|
| chdir: changes the working      | directory . . . . .                        | chdir(S)       |
| chroot: changes the root        | directory . . . . .                        | chroot(S)      |
| uuclean: uucp spool             | directory clean-up . . . . .               | uuclean(ADM)   |
| lc: lists                       | directory contents in columns . . . . .    | lc(C)          |
| dir: format of a                | directory . . . . .                        | dir(F)         |
| file getdents: read             | directory entries and put in a . . . . .   | getdents(S)    |
| dirent: file-system-independent | directory entry . . . . .                  | dirent(F)      |
| dirent: filesystem independent  | directory entry . . . . .                  | dirent(F)      |
| unlink: removes                 | directory entry . . . . .                  | unlink(S)      |
| chroot: changes root            | directory for command . . . . .            | chroot(ADM)    |
| uucico: scan the spool          | directory for work . . . . .               | uucico(C)      |
| mkdir: makes a                  | directory . . . . .                        | mkdir(C)       |
| mkdir: creates a new            | directory . . . . .                        | mkdir(DOS)     |
| mkdir: make a                   | directory . . . . .                        | mkdir(S)       |
| mmdir: moves a                  | directory . . . . .                        | mmdir(C)       |
| pwd: prints working             | directory name . . . . .                   | pwd(C)         |
| basename: removes               | directory names from pathnames . . . . .   | basename(C)    |
| closedir: performs              | directory operations . . . . .             | directory(S)   |
| ordinary file mknod: makes a    | directory, or a special or . . . . .       | mknod(S)       |
| dirname: delivers               | directory part of pathname . . . . .       | dirname(C)     |
| rename: renames a file or       | directory . . . . .                        | rename(DOS)    |
| rmdir: deletes a                | directory . . . . .                        | rmdir(DOS)     |
| rmdir: remove a                 | directory . . . . .                        | rmdir(S)       |
| temporary files in the current  | directory /and deletes all . . . . .       | rmtmp(DOS)     |
| access permissions of a file or | directory chmod: changes the . . . . .     | chmod(C)       |
| default program information     | directory default: . . . . .               | default(F)     |
| the pathname of current working | directory getcwd: get . . . . .            | getcwd(S)      |
| information about contents of   | directory l: lists . . . . .               | l(C)           |
| directory entry                 | dirent: filesystem independent . . . . .   | dirent(F)      |
| directory entry                 | dirent: file-system-independent . . . . .  | dirent(F)      |
| of pathname                     | dirname: delivers directory part . . . . . | dirname(C)     |
|                                 | dis: object code disassembler . . . . .    | dis(CP)        |
| t_unbind:                       | disable a transport endpoint . . . . .     | t_unbind(NSL)  |
| session chg_audit: enables and  | disable auditing for the next . . . . .    | chg_audit(ADM) |
| printers                        | disable: turns off terminals and . . . . . | disable(C)     |
| emunmap:                        | disables mapping on a channel . . . . .    | emunmap(K)     |
| acct: enables or                | disables process accounting . . . . .      | acct(S)        |
| dis: object code                | disassembler . . . . .                     | dis(CP)        |
| the queue ev_flush:             | discard all events currently in . . . . .  | ev_flush(S)    |
| type, modes, speed, and line    | discipline /set terminal . . . . .         | ugetty(ADM)    |
| type, modes, speed, and line    | discipline /Sets terminal . . . . .        | getty(M)       |
| t_snddis: send user-initiated   | disconnect request . . . . .               | t_snddis(NSL)  |
| retrieve information from       | disconnect t_rcvdis: . . . . .             | t_rcvdis(NSL)  |
| attributes of files and/        | discr: check discretionary . . . . .       | discr(S)       |
| files and programs discr: check | discretionary attributes of . . . . .      | discr(S)       |
| diskusg: generate               | disk accounting data by user ID . . . . .  | diskusg(ADM)   |
| cmchk: reports hard             | disk block size . . . . .                  | cmchk(C)       |
| df: report number of free       | disk blocks . . . . .                      | df(C)          |
| dparam: displays/changes hard   | disk characteristics . . . . .             | dparam(ADM)    |
| hd: internal hard               | disk drive . . . . .                       | hd(HW)         |
| track/ badtrk: scans fixed      | disk for flaws and creates bad . . . . .   | badtrk(ADM)    |



|                                  |                                            |                   |
|----------------------------------|--------------------------------------------|-------------------|
| fdisk: maintain                  | disk partitions . . . . .                  | fdisk(ADM)        |
| dtype: determines                | disk type . . . . .                        | dtype(C)          |
| du: summarizes                   | disk usage . . . . .                       | du(C)             |
| floppy disks diskcp,             | diskcmp: copies or compares . . . . .      | diskcp(C)         |
| compares floppy disks            | diskcp, diskcmp: copies or . . . . .       | diskcp(C)         |
| format: format floppy            | disks . . . . .                            | format(C)         |
| copies or compares floppy        | disks diskcp, diskcmp: . . . . .           | diskcp(C)         |
| request to a device's queue      | disksort: adds a block I/O . . . . .       | disksort(K)       |
| accounting data by user ID       | diskusg: generate disk . . . . .           | diskusg(ADM)      |
| umount:                          | dismounts a file structure . . . . .       | umount(ADM)       |
| zcat:                            | display a stored file . . . . .            | compress(C)       |
| vedit: invokes a screen-oriented | display editor vi, view, . . . . .         | vi(C)             |
| displaypkg:                      | display installed packages . . . . .       | displaypkg(ADM)   |
| count profile data lprof:        | display line-by-line execution . . . . .   | lprof(CP)         |
| vidinitscreen, vidmap,/ video:   | DISPLAYED, viddoio, . . . . .              | video(K)          |
| packages                         | displaypkg: display installed . . . . .    | displaypkg(ADM)   |
| configuration data base cmos:    | displays and sets the . . . . .            | cmos(HW)          |
| message printcfg:                | displays driver initialization . . . . .   | printcfg(K)       |
| cat: concatenates and            | displays files . . . . .                   | cat(C)            |
| format hd:                       | displays files in hexadecimal . . . . .    | hd(C)             |
| od:                              | displays files in octal format . . . . .   | od(C)             |
| system activity uptime:          | displays information about . . . . .       | uptime(C)         |
| is on the system and what w:     | displays information about who . . . . .   | w(C)              |
| system cmn_err:                  | displays message or panics the . . . . .   | cmn_err(K)        |
| prof:                            | displays profile data . . . . .            | prof(CP)          |
| prof:                            | displays profile data . . . . .            | prof(XNX)         |
| executable binary files hdr:     | displays selected parts of . . . . .       | hdr(CP)           |
| device numbers/ majorsinuse:     | displays the list of major . . . . .       | majorsinuse(ADM)  |
| currently/ vectorsinuse:         | displays the list of vectors . . . . .     | vectorsinuse(ADM) |
| characteristics dparam:          | displays/changes hard disk . . . . .       | dparam(ADM)       |
| mail: sends, reads or            | disposes of mail . . . . .                 | mail(C)           |
| cabs: determines Euclidean       | distance hypot, . . . . .                  | hypot(S)          |
| lcong48: generates uniformly     | distributed srand48, seed48, . . . . .     | drand48(S)        |
|                                  | div: divides integers . . . . .            | div(DOS)          |
|                                  | div: divides integers . . . . .            | div(S)            |
|                                  | div: divides integers . . . . .            | div(DOS)          |
|                                  | div: divides integers . . . . .            | div(S)            |
|                                  | ldiv: divides long integers . . . . .      | ldiv(DOS)         |
|                                  | ldiv: divides long integers . . . . .      | ldiv(S)           |
|                                  | divvy -b block_device -c c/ . . . . .      | divvy(ADM)        |
| records for subsystem events     | dlvr_audit: produce audit . . . . .        | dlvr_audit(ADM)   |
| dma_alloc: allocates a           | DMA channel . . . . .                      | dma_alloc(K)      |
| releases previously allocated    | DMA channel dma_relse: . . . . .           | dma_relse(K)      |
| transfer dma_param: sets up a    | DMA controller chip for DMA . . . . .      | dma_param(K)      |
| dma_start: queues                | DMA request . . . . .                      | dma_start(K)      |
| dma_breakup: sizes               | DMA request into 512-byte blocks . . . . . | dma_breakup(K)    |
| bytes not transferred during a   | DMA request /the number of . . . . .       | dma_resid(K)      |
| dma_enable: begins               | DMA transfer . . . . .                     | dma_enable(K)     |
| up a DMA controller chip for     | DMA transfer dma_param: sets . . . . .     | dma_param(K)      |
| channel                          | dma_alloc: allocates a DMA . . . . .       | dma_alloc(K)      |
| into 512-byte blocks             | dma_breakup: sizes DMA request . . . . .   | dma_breakup(K)    |

|                                                                   |                                      |                   |
|-------------------------------------------------------------------|--------------------------------------|-------------------|
| controller chip for DMA/<br>allocated DMA channel                 | dma_enable: begins DMA transfer      | dma_enable(K)     |
| bytes not transferred during a/<br>object downloader for the 5620 | dma_param: sets up a DMA . . .       | dma_param(K)      |
| conversions/ dieeetomshin,                                        | dma_rlse: releases previously . . .  | dma_rlse(K)       |
| acctsh: chargefee, ckpacct,                                       | dma_resid: returns the number of . . | dma_resid(K)      |
| whodo: determines who is                                          | dma_start: queues DMA request . .    | dma_start(K)      |
| promain: restrict the execution                                   | DMD terminal wtinit: . . . . .       | wtinit(ADM)       |
| intro: introduction to                                            | dmsbintoiee: perform . . . . .       | dieeetomshin(DOS) |
| dosexterr: gets                                                   | dodisk, lastlogin, monacct/ . . .    | acctsh(ADM)       |
| dosls, dosrm, dosrmdir: access                                    | doing what . . . . .                 | whodo(C)          |
| bdos: invokes a                                                   | domain of a program . . . . .        | promain(M)        |
| intdos: invokes a                                                 | dOS cross development functions . .  | intro(DOS)        |
| intdosx: invokes a                                                | dOS error messages . . . . .         | dosexter(DOS)     |
| extended error information                                        | dOS files . . . . .                  | dos(C)            |
| messages                                                          | dOS system call . . . . .            | bdos(DOS)         |
| linker                                                            | dOS system call . . . . .            | intdos(DOS)       |
| dOS files                                                         | dOS system call . . . . .            | intdosx(DOS)      |
| files dosls,                                                      | dosexterr: gets and stores . . .     | dosexterr(DOS)    |
| dosls, dosrm,                                                     | dosexterr: gets DOS error . . .      | dosexter(DOS)     |
| dosrmdir: access                                                  | dosld: XENIX to MS-DOS cross . .     | dosld(CP)         |
| dosrmdir: access DOS files                                        | dosls, dosrm, dosrmdir: access . .   | dos(C)            |
| dosrmdir: access DOS files                                        | dosrm, dosrmdir: access DOS . .      | dos(C)            |
| /atof: converts a string to a                                     | double-precision number . . . . .    | strtod(S)         |
| DMD/ wtinit: object                                               | downloader for the 5620 . . . . .    | wtinit(ADM)       |
| disk characteristics                                              | dparam: displays/changes hard . .    | dparam(ADM)       |
| graph:                                                            | draw a graph . . . . .               | graph(ADM)        |
| hd: internal hard disk                                            | drive . . . . .                      | hd(HW)            |
| swaps default boot floppy                                         | drive fdswap: . . . . .              | fdswap(ADM)       |
| administration/ atcronsh: menu                                    | driven at and cron . . . . .         | atcronsh(ADM)     |
| utility auditsh: menu                                             | driven audit administration . . .    | auditsh(ADM)      |
| utility backupsh: menu                                            | driven backup administration . . .   | backupsh(ADM)     |
| administration/ lpsh: menu                                        | driven lp print service . . . . .    | lpsh(ADM)         |
| utility sysadmsh: menu                                            | driven system administration . . .   | sysadmsh(ADM)     |
| file mvdevice: video                                              | driver backend configuration . . .   | mvdevice(F)       |
| delete, update, or get device                                     | driver configuration data /add, . .  | idinstall(ADM)    |
| vidunmap: supports video adapter                                  | driver development /vidumapinit, .   | video(K)          |
| object module routines: finds                                     | driver entry points in a driver . .  | routines(ADM)     |
| terminals xt: multiplexed tty                                     | driver for AT&T windowing . . .      | xt(HW)            |
| printcfg: displays                                                | driver initialization message . . .  | printcfg(K)       |
| ttiocom: interpret tty                                            | driver I/O control commands . . .    | ttiocom(K)        |
| finds driver entry points in a                                    | driver object module routines: . .   | routines(ADM)     |
| xtt: extract and print xt                                         | driver packet traces . . . . .       | xtt(ADM)          |
| ttxput, ttyflush, ttywait: tty                                    | driver routines /ttwrite, . . . . .  | tty(K)            |
| xts: extract and print xt                                         | driver statistics . . . . .          | xts(ADM)          |
| sxt: pseudo-device                                                | driver . . . . .                     | sxt(M)            |
| protocol used by xt(HW)                                           | driver /multiplexed channels . . .   | xtproto(M)        |
| physck: raw I/O for block                                         | drivers physio, . . . . .            | physio(K)         |
| term: terminal                                                    | driving tables for nroff . . . . .   | term(F)           |
|                                                                   | dtype: determines disk type . . .    | dtype(C)          |
|                                                                   | du: summarizes disk usage . . . .    | du(C)             |
| authaudit: produce audit records                                  | due to authentication events . . .   | authaudit(S)      |



|                                   |                                  |                |
|-----------------------------------|----------------------------------|----------------|
| object file                       | dump: dump selected parts of an  | dump(CP)       |
| file dump:                        | dump selected parts of an object | dump(CP)       |
| backup: incremental               | dump tape format                 | backup(F)      |
| file tapedump:                    | dumps magnetic tape to output    | tapedump(C)    |
| file descriptor                   | dup, dup2: duplicates an open    | dup(S)         |
| descriptor                        | dup2: duplicate an open file     | dup2(S)        |
| descriptor dup,                   | dup2: duplicates an open file    | dup(S)         |
| descriptor dup2:                  | duplicate an open file           | dup2(S)        |
| descriptor dup, dup2:             | duplicates an open file          | dup(S)         |
| emdupmap:                         | duplicates channel mapping       | emdupmap(K)    |
| _freect: counts available         | dynamic memory                   | _freect(DOS)   |
|                                   | echo: echoes arguments           | echo(C)        |
| getche: gets and                  | echoes a character               | getche(DOS)    |
| echo:                             | echoes arguments                 | echo(C)        |
| output conversions                | ecvt, fcvt, gcv: performs        | ecvt(S)        |
|                                   | ed: invokes the text editor      | ed(C)          |
| program end, etext,               | edata: last locations in         | end(S)         |
| for casual users)                 | edit: text editor (variant of ex | edit(C)        |
| sact: prints current SCCS file    | editing activity                 | sact(CP)       |
| ed: invokes the text              | editor                           | ed(C)          |
| ex: invokes a text                | editor                           | ex(C)          |
| ld: invokes the link              | editor                           | ld(CP)         |
| ld: invokes the link              | editor                           | ld(M)          |
| ld: invokes the link              | editor                           | ld(XNX)        |
| format of assembler and link      | editor output a.out:             | a.out(F)       |
| the stream                        | editor sed: invokes              | sed(C)         |
| users) edit: text                 | editor (variant of ex for casual | edit(C)        |
| a screen-oriented display         | editor /view, vedit: invokes     | vi(C)          |
| effective user, real group, and   | effective group IDs /real user,  | getuid(S)      |
| /getgid, getegid: gets real user, | effective user, real group, and/ | getuid(S)      |
| color, monochrome,                | ega, /tty[01-n],                 | screen(HW)     |
| for a pattern grep,               | egrep, fgrep: searches a file    | grep(C)        |
| mapping                           | emdupmap: duplicates channel     | emdupmap(K)    |
| i286emul:                         | emulate 80286                    | i286emul(C)    |
| x286emul:                         | emulate XENIX 80286              | x286emul(C)    |
| channel                           | emunmap: disables mapping on a   | emunmap(K)     |
| line printers                     | enable: turns on terminals and   | enable(C)      |
| the next session chg_audit:       | enables and disable auditing for | chg_audit(ADM) |
| accounting acct:                  | enables or disables process      | acct(S)        |
| transmission via mail uencode:    | encode a binary file for         | uencode(C)     |
| crypt:                            | encode/decode                    | crypt(C)       |
| crypt: password and file          | encryption functions             | crypt(S)       |
| makekey: generates an             | encryption key                   | makekey(M)     |
| locations in program              | end, etext, edata: last          | end(S)         |
| /getdvagname, setdvagname,        | enddvagname, putdvagname,/       | getdvagname(S) |
| /getgrgid, getgrnam, setgrnt,     | endgrnt: get group file entry    | getgrnt(S)     |
| eof: determines                   | end-of-file                      | eof(DOS)       |
| t_close: close a transport        | endpoint                         | t_close(NSL)   |
| t_open: establish a transport     | endpoint                         | t_open(NSL)    |
| t_unbind: disable a transport     | endpoint                         | t_unbind(NSL)  |
| bind an address to a transport    | endpoint t_bind:                 | t_bind(NSL)    |



|                                   |                                             |                 |
|-----------------------------------|---------------------------------------------|-----------------|
| the current event on a transport  | endpoint t_look: look at . . . . .          | t_look(NSL)     |
| manage options for a transport    | endpoint t_optmgmt: . . . . .               | t_optmgmt(NSL)  |
| /getprcmnam, setprcmnt,           | endprcmnt, putprcmnam:/ . . . . .           | getprcmnt(S)    |
| /getprdfnam, setprdfnt,           | endprdfnt, putprdfnam:/ . . . . .           | getprdfnt(S)    |
| /getprfinam, setprfient,          | endprfient, putprfinam:/ . . . . .          | getprfient(S)   |
| /getprpwnam, setprpwent,          | endprpwent, putprpwnam:/ . . . . .          | getprpwent(S)   |
| /getprtncnam, setprtcent,         | endprtcent, putprtncnam:/ . . . . .         | getprtcent(S)   |
| /getpwuid, getpwnam, setpwent,    | endpwent: gets password file/ . . . . .     | getpwent(S)     |
| error longjmp:                    | ends current system call with . . . . .     | longjmp(K)      |
| utmp file entry                   | endutent, utmpname: accesses . . . . .      | getut(S)        |
| submit: MMDf mail                 | enqueueer . . . . .                         | submit(ADM)     |
| dblock: lock the                  | entire Authentication database . . . . .    | dblock(S)       |
| getdents: read directory          | entries and put in a file . . . . .         | getdents(S)     |
| defopen, defread: reads default   | entries . . . . .                           | defopen(S)      |
| xlist, fxlist: gets name list     | entries from files . . . . .                | xlist(S)        |
| nlist: gets                       | entries from name list . . . . .            | nlist(S)        |
| linenum: line number              | entries in a common object file . . . . .   | linenum(F)      |
| /ldlitem: manipulate line number  | entries of a common object file/ . . . . .  | ldlread(S)      |
| /ldnlseek: seek to line number    | entries of a section of a common/ . . . . . | ldlseek(S)      |
| /ldnrseek: seek to relocation     | entries of a section of a common/ . . . . . | ldrseek(S)      |
| wtmp: formats of utmp and wtmp    | entries utmp, . . . . .                     | utmp(F)         |
| /the index of a symbol table      | entry of a common object file . . . . .     | ldtbindex(S)    |
| /read an indexed symbol table     | entry of a common object file . . . . .     | ldtbread(S)     |
| module routines: finds driver     | entry points in a driver object . . . . .   | routines(ADM)   |
| putpwent: writes a password file  | entry . . . . .                             | putpwent(S)     |
| the address of the next heap      | entry structure /returns . . . . .          | _fheapwalk(DOS) |
| relogin: rename login             | entry to show current layer . . . . .       | relogin(ADM)    |
| unlink: removes directory         | entry . . . . .                             | unlink(S)       |
| device assignment database        | entry /copydvagent: manipulate . . . . .    | getdvagent(S)   |
| system independent directory      | entry dirent: file . . . . .                | dirent(F)       |
| manipulate file control database  | entry /endprfient, putprfinam: . . . . .    | getprfient(S)   |
| utmpname: accesses utmp file      | entry endutent, . . . . .                   | getut(S)        |
| directory                         | entry /file-system-independent . . . . .    | dirent(F)       |
| endgrent: get group file          | entry /getgrnam, setgrent, . . . . .        | getgrent(S)     |
| endpwent: gets password file      | entry /getpwnam, setpwent, . . . . .        | getpwent(S)     |
| command control database          | entry /putprcmnam: manipulate . . . . .     | getprcmnt(S)    |
| default control database          | entry /putprdfnam: manipulate . . . . .     | getprdfnt(S)    |
| protected password database       | entry /putprpwnam: manipulate . . . . .     | getprpwent(S)   |
| terminal control database         | entry /putprtncnam: manipulate . . . . .    | getprtcent(S)   |
| common object file symbol table   | entry /retrieve symbol name for . . . . .   | ldgetname(S)    |
| command execution                 | env: sets environment for . . . . .         | env(C)          |
| profile: sets up an               | environ: the user environment . . . . .     | environ(M)      |
| /fpsetsticky: IEEE floating point | environment at login time . . . . .         | profile(M)      |
| environ: the user                 | environment control . . . . .               | fpgetround(S)   |
| execution env: sets               | environment . . . . .                       | environ(M)      |
| getenv: gets value for            | environment for command . . . . .           | env(C)          |
| putenv: changes or adds value to  | environment name . . . . .                  | getenv(S)       |
| TZ: time zone                     | environment . . . . .                       | putenv(S)       |
| commands performed for multiuser  | environment variable . . . . .              | tz(M)           |
| set or read international         | environment rc2: run . . . . .              | rc2(ADM)        |
|                                   | environment setlocale: . . . . .            | setlocale(S)    |

|                                  |                                            |                 |
|----------------------------------|--------------------------------------------|-----------------|
|                                  | eof: determines end-of-file . . . . .      | eof(DOS)        |
| complementary error function     | erf, erfc: error function and . . . . .    | erf(S)          |
| complementary error/ erf,        | erfc: error function and . . . . .         | erf(S)          |
| perror, sys_errlist, sys_nerr,   | errno: sends system error/ . . . . .       | perror(S)       |
| seterror: sets u.u_error with    | error code . . . . .                       | seterror(K)     |
| error function erf, erfc:        | error function and complementary           | erf(S)          |
| error function and complementary | error function erf, erfc: . . . . .        | erf(S)          |
| t_rcvuderr: receive a unit data  | error indication . . . . .                 | t_rcvuderr(NSL) |
| gets and stores extended         | error information dosexterr: . . . . .     | dosexterr(DOS)  |
| device                           | error: kernel error output . . . . .       | error(M)        |
| strclean: STREAMS                | error logger cleanup program . . . . .     | strclean(ADM)   |
| strerr: STREAMS                  | error logger daemon . . . . .              | strerr(ADM)     |
| source mkstr: creates an         | error message file from C . . . . .        | mkstr(CP)       |
| deverr: prints a device          | error message on the console . . . . .     | deverr(K)       |
| routine call/ strerror: gets     | error message pointer from last . . . . .  | strerror(DOS)   |
| routine call/ strerror: gets     | error message pointer from last . . . . .  | strerror(S)     |
| t_error: produce                 | error message . . . . .                    | t_error(NSL)    |
| dosexterr: gets DOS              | error messages . . . . .                   | dosexterr(DOS)  |
| sys_nerr, errno: Sends system    | error messages /sys_errlist, . . . . .     | perror(S)       |
| services, library routines and   | error numbers /system . . . . .            | Intro(S)        |
| error: kernel                    | error output device . . . . .              | error(M)        |
| fsave: interactive,              | error-checking filesystem backup . . . . . | fsave(ADM)      |
| pointer from last routine call   | error /gets error message . . . . .        | strerror(DOS)   |
| pointer from last routine call   | error /gets error message . . . . .        | strerror(S)     |
| matherr:                         | error-handling function . . . . .          | matherr(S)      |
| ends current system call with    | error longjmp: . . . . .                   | longjmp(K)      |
| hashcheck: finds spelling        | errors /hashmake, spellin, . . . . .       | spell(C)        |
| another transport/ t_connect:    | establish a connection with . . . . .      | t_connect(NSL)  |
| t_open:                          | establish a transport endpoint . . . . .   | t_open(NSL)     |
| terminal line connection dial:   | establishes an out-going . . . . .         | dial(S)         |
| setmnt:                          | establishes /etc/mnttab table . . . . .    | setmnt(ADM)     |
| setmnt: establishes              | /etc/mnttab table . . . . .                | setmnt(ADM)     |
| program end,                     | etext, edata: last locations in . . . . .  | end(S)          |
| hypot, cabs: determines          | euclidean distance . . . . .               | hypot(S)        |
| expression expr:                 | evaluates arguments as an . . . . .        | expr(C)         |
| contains an event                | ev_block: wait until the queue . . . . .   | ev_block(S)     |
| and all associated devices       | ev_close: close the event queue . . . . .  | ev_close(S)     |
| events currently in the queue    | ev_count: returns the number of . . . . .  | ev_count(S)     |
| ev_read: read the next           | event in the queue . . . . .               | ev_read(S)      |
| include/exclude devices for      | event input ev_gindev: . . . . .           | ev_gindev(S)    |
| ev_init: invokes the             | event manager . . . . .                    | ev_init(S)      |
| ev_getemask: return the current  | event mask . . . . .                       | ev_getemask(S)  |
| ev_getemask: return the current  | event mask . . . . .                       | ev_gtemask(S)   |
| ev_setemask: sets                | event mask . . . . .                       | ev_setemask(S)  |
| ev_setemask: Sets                | event mask . . . . .                       | ev_stemask(S)   |
| ev_pop: pop the next             | event off the queue . . . . .              | ev_pop(S)       |
| t_look: look at the current      | event on a transport endpoint . . . . .    | t_look(NSL)     |
| devices ev_close: close the      | event queue and all associated . . . . .   | ev_close(S)     |
| ev_suspend: suspends an          | event queue . . . . .                      | ev_suspend(S)   |
| ev_suspend: Suspends an          | event queue . . . . .                      | ev_susp(S)      |
| ev_open: opens an                | event queue for input . . . . .            | ev_open(S)      |



|                                  |                                  |                          |                 |
|----------------------------------|----------------------------------|--------------------------|-----------------|
| a list of devices feeding an     | event queue                      | ev_getdev: gets          | ev_getdev(S)    |
| wait until the queue contains an | event                            | ev_block:                | ev_block(S)     |
| ev_count: returns the number of  | events currently in the queue    |                          | ev_count(S)     |
| ev_flush: discard all            | events currently in the queue    |                          | ev_flush(S)     |
| records due to authentication    | events                           | authaudit: produce audit | authaudit(S)    |
| audit records for subsystem      | events                           | dlvr_audit: produce      | dlvr_audit(ADM) |
| currently in the queue           | ev_flush: discard all events     |                          | ev_flush(S)     |
| devices feeding an event queue   | ev_getdev: gets a list of        |                          | ev_getdev(S)    |
| event mask                       | ev_getemask: return the current  |                          | ev_getemask(S)  |
| event mask                       | ev_getemask: return the current  |                          | ev_gtemask(S)   |
| devices for event input          | ev_gindev: include/exclude       |                          | ev_gindev(S)    |
| manager                          | ev_init: invokes the event       |                          | ev_init(S)      |
| for input                        | ev_open: opens an event queue    |                          | ev_open(S)      |
| the queue                        | ev_pop: pop the next event off   |                          | ev_pop(S)       |
| the queue                        | ev_read: read the next event in  |                          | ev_read(S)      |
| queue                            | ev_resume: restart a suspended   |                          | ev_resume(S)    |
| queue                            | ev_setemask: sets event mask     |                          | ev_setemask(S)  |
| queue                            | ev_setemask: Sets event mask     |                          | ev_stemask(S)   |
| queue                            | ev_suspend: suspends an event    |                          | ev_suspend(S)   |
| queue                            | ev_suspend: Suspends an event    |                          | ev_susp(S)      |
| edit: text editor (variant of    | ex for casual users)             |                          | edit(C)         |
| cscope: interactively            | ex: invokes a text editor        |                          | ex(C)           |
| signals                          | examine a C program              |                          | cscope(CP)      |
| sigprocmask:                     | examine and change blocked       |                          | sigprocmask(S)  |
| sigaction:                       | examine and change signal action |                          | sigaction(S)    |
| sigpending:                      | examine pending signals          |                          | sigpending(S)   |
| authentication/ integrity:       | examine system files against     |                          | integrity(ADM)  |
| crash:                           | examine system images            |                          | crash(ADM)      |
| pax: portable archive            | exchange                         |                          | pax(C)          |
| execv, execve, execvp, execvpe:/ | execl, execl, execlp, execlpe,   |                          | exec(DOS)       |
| execlp, execvp: executes a/      | execl, execv, execl, execve,     |                          | exec(S)         |
| execve, execvp, execvpe:/ execl, | execl, execlp, execlpe, execv,   |                          | exec(DOS)       |
| executes a file                  | execl, execlp, execlp, execvp:   |                          | exec(S)         |
| execvp, execvpe:/ execl, execl,  | execlp, execlpe, execv, execve,  |                          | exec(DOS)       |
| execl, execv, execl, execve,     | execlp, execvp: executes a file  |                          | exec(S)         |
| execvpe:/ execl, execl, execlp,  | execlpe, execv, execve, execvp,  |                          | exec(DOS)       |
| executable                       | execseg: makes a data region     |                          | execseg(S)      |
| fixhdr: changes                  | executable binary file headers   |                          | fixhdr(C)       |
| hdr: displays selected parts of  | executable binary files          |                          | hdr(CP)         |
| execseg: makes a data region     | executable                       |                          | execseg(S)      |
| execvp, execvpe: load and        | execute a process /execve,       |                          | exec(DOS)       |
| untimeout: schedules a time to   | execute a routine                | timeout,                 | timeout(K)      |
| regcmp, regex: compile and       | execute regular expression       |                          | regcmp(S)       |
| execl, execve, execlp, execvp:   | executes a file                  | execl, execv,            | exec(S)         |
| system:                          | executes a shell command         |                          | system(S)       |
| int86:                           | executes an interrupt            |                          | int86(DOS)      |
| int86x:                          | executes an interrupt            |                          | int86x(DOS)     |
| command system:                  | executes an operating system     |                          | system(DOS)     |
| UNIX uux:                        | executes command on remote       |                          | uux(C)          |
| time at, batch:                  | executes commands at a later     |                          | at(C)           |
| times cron:                      | executes commands at specified   |                          | cron(C)         |



|                                      |                                  |               |
|--------------------------------------|----------------------------------|---------------|
| UNIX system remote:                  | executes commands on a remote    | remote(C)     |
| xargs: constructs and                | executes commands                | xargs(C)      |
| regex, regcmp: compiles and          | executes regular expressions     | regex(S)      |
| raise: sends a signal to the         | executing program                | raise(DOS)    |
| lprof: display line-by-line          | execution count profile data     | lprof(CP)     |
| promain: restrict the                | execution domain of a program    | promain(M)    |
| nap: Suspends                        | execution for a short interval   | nap(S)        |
| sleep: Suspends                      | execution for an interval        | sleep(C)      |
| sleep: Suspends                      | execution for an interval        | sleep(S)      |
| delay: delays process                | execution for specified time     | delay(K)      |
| monitor: prepares                    | execution profile                | monitor(S)    |
| raise: send signal sig to            | execution program                | raise(S)      |
| profil: creates an                   | execution time profile           | profil(S)     |
| sets environment for command         | execution env:                   | env(C)        |
| execvp: executes a file              | execv, execle, execve, execlp,   | exec(S)       |
| execl, execl, execlp, execlpe,       | execv, execve, execvp, execvpe:/ | exec(DOS)     |
| a file                               | execve, execlp, execvp: executes | exec(S)       |
| execl, execv, execl,                 | execve, execvp, execvpe: load/   | exec(DOS)     |
| /execl, execlp, execlpe, execv,      | execvp: executes a file          | exec(S)       |
| execv, execl, execve, execlp,        | execvp, execvpe: load and/       | exec(DOS)     |
| /execlp, execlpe, execv, execve,     | execvpe: load and execute a/     | exec(DOS)     |
| execlp, execv, execve, execvp,       | existing file                    | link(S)       |
| link: links a new filename to an     | existing one creat: creates      | creat(DOS)    |
| a new file or overwrites an          | existing one creat: creates      | creat(S)      |
| a new file or rewrites an            | exists and is accessible         | access(DOS)   |
| /determines whether a file           | exit, _exit: terminates a        | exit(S)       |
| process                              | _exit: terminates a process      | exit(S)       |
| exit,                                | exit: terminates the calling     | exit(DOS)     |
| process                              | exit value                       | false(C)      |
| false: returns with a nonzero        | exit value                       | true(C)       |
| true: returns with a zero            | exp, log, pow, sqrt, log10:      | exp(S)        |
| performs exponential,/               | _expand: changes the size of a   | _expand(DOS)  |
| previously allocated memory/         | expands files                    | pack,         |
| pcat, unpack: compresses and         | expedited data over a connection | pack(C)       |
| t_snd: send data or                  | expedited data sent over a/      | t_snd(NSL)    |
| t_rcv: receive data or               | exponential, logarithm, power,/  | t_rcv(NSL)    |
| /log, pow, sqrt, log10: performs     | exponent /Splits floating-point  | exp(S)        |
| number into a mantissa and an        | expr: evaluates arguments as an  | frexp(S)      |
| expression                           | expression compile and match     | expr(C)       |
| regex: regular                       | expression                       | regexp(S)     |
| expr: evaluates arguments as an      | expression regcmp, regex:        | expr(C)       |
| compile and execute regular          | expressions                      | regcmp(S)     |
| regcmp: compiles regular             | expressions regex, regcmp:       | regcmp(CP)    |
| compiles and executes regular        | extended error information       | regex(S)      |
| dosxterr: gets and stores            | extra blank lines from a file    | dosxterr(DOS) |
| rmb: remove                          | extract and print xt driver      | rmb(M)        |
| statistics                           | extract and print xt driver      | xts(ADM)      |
| xts: packet traces                   | extracts strings from C          | xtt(ADM)      |
| xtt: programs                        | fabs, ceil, fmod: performs       | xstr(CP)      |
| xstr: absolute value, floor,/ floor, | facilities /Reports the status   | floor(S)      |
| of inter-process communication       | factor: factor a number          | floor(S)      |
| factor:                              |                                  | ipcs(ADM)     |
|                                      |                                  | factor(C)     |

|                                   |                                                   |                  |
|-----------------------------------|---------------------------------------------------|------------------|
|                                   | factor: factor a number . . . . .                 | factor(C)        |
| exit value                        | false: returns with a nonzero . . .               | false(C)         |
| abort: generates an IOT           | fault . . . . .                                   | abort(S)         |
| streams                           | fclose, fcloseall: closes . . . . .               | fclose(DOS)      |
| flushes a stream                  | fclose, fflush: closes or . . . . .               | fclose(S)        |
| fclose,                           | fcloseall: closes streams . . . . .               | fclose(DOS)      |
|                                   | fcntl: controls open files . . . . .              | fcntl(S)         |
|                                   | fcntl: file control options . . . . .             | fcntl(M)         |
| conversions                       | ecvt, fcvt, gcvt: performs output . . . .         | ecvt(S)          |
| ecvt,                             | fdisk: maintain disk partitions . . .             | fdisk(ADM)       |
| fopen, freopen,                   | fdopen: opens a stream . . . . .                  | fopen(S)         |
| floppy drive                      | fdswap: Swaps default boot . . . .                | fdswap(ADM)      |
| /to machine related miscellaneous | features and files . . . . .                      | Intro(HW)        |
| introduction to miscellaneous     | features and files intro: . . . . .               | Intro(M)         |
| /Gets a list of devices           | feeding an event queue . . . . .                  | ev_getdev(S)     |
| determines stream/                | feof, clearerr, fileno: . . . . .                 | ferror(S)        |
| determines stream status          | ferror, feof, clearerr, fileno: . . .             | ferror(S)        |
| nextkey: performs/                | fetch, store, delete, firstkey, . . .             | dbm(S)           |
| dbminit,                          | fflush: closes or flushes a . . . . .             | fclose(S)        |
| stream                            | fclose, _ffree, _nfree: deallocate . . . .        | free(DOS)        |
| memory blocks                     | free, fgetc, fgetchar: gets a . . . . .           | fgetc(DOS)       |
| free,                             | fgetc, getw: gets character or . . . . .          | getc(S)          |
| character from a stream           | fgetc, getw: gets character from . . .            | fgetc(DOS)       |
| word from a/                      | fgetc, getw: gets character or . . . . .          | getc(S)          |
| getc, getchar,                    | fgetc, getw: gets character from . . .            | fgetc(DOS)       |
| a stream                          | fgetc, getw: gets character or . . . . .          | getc(S)          |
| fgetc,                            | fgetc, getw: gets character from . . .            | fgetc(DOS)       |
| current value of a stream's/      | fgetc, getw: gets character or . . . . .          | getc(S)          |
| current value of a stream's/      | fgetc, getw: gets character from . . .            | fgetc(DOS)       |
| stream                            | fgetc, getw: gets character or . . . . .          | getc(S)          |
| gets,                             | fgetc, getw: gets character from . . .            | fgetc(DOS)       |
| pattern                           | grep, egrep, fgrep: Searches a file for a . . . . | grep(C)          |
| minimal consistency check on/     | _fheapchk, _nheapchk: performs a . . .            | _fheapchk(DOS)   |
| the address of the next heap/     | _fheapwalk, _nheapwalk: returns . . .             | _fheapwalk(DOS)  |
| perform conversions between/      | fiectomsbin, fmsbintoieee: . . . . .              | fiectomsbin(DOS) |
| fields: return status based on    | fields of authentication/ . . . . .               | fields(S)        |
| cut: cuts out selected            | fields of each line of a file . . . . .           | cut(C)           |
| fields of authentication/         | fields: return status based on . . . .            | fields(S)        |
| routines                          | fieldtype: FIELDTYPE library . . . . .            | fieldtype(S)     |
| fieldtype:                        | FIELDTYPE library routines . . . . .              | fieldtype(S)     |
| times                             | utime: Sets file access and modification . . . .  | utime(S)         |
| utime: Sets                       | file access routines . . . . .                    | ldfcn(F)         |
| ldfcn: common object              | file archives in and out . . . . .                | cpio(C)          |
| cpio: copies                      | file . . . . .                                    | chmod(S)         |
| chmod: changes mode of a          | file . . . . .                                    | chsize(S)        |
| chsize: changes the size of a     | file comment section . . . . .                    | mcs(CP)          |
| mcs: manipulate the object        | file . . . . .                                    | compress(C)      |
| uncompress: uncompress a stored   | file . . . . .                                    | compress(C)      |
| zcat: display a stored            | file control database entry . . . . .             | getprfient(S)    |
| /putprfinam: manipulate           | fcntl: file control options . . . . .             | fcntl(M)         |
| fcntl:                            | file converter . . . . .                          | conv(CP)         |
| conv: common object               | file copy uuto, . . . . .                         | uuto(C)          |
| uupick: public UNIX-to-UNIX       | file . . . . .                                    | core(F)          |
| core: format of core image        | file . . . . .                                    | cprs(CP)         |
| cprs: compress a common object    | file creation mask . . . . .                      | umask(S)         |
| umask: Sets and gets              | file . . . . .                                    | crontab(C)       |
| crontab: user crontab             |                                                   |                  |



|                                  |                                  |               |
|----------------------------------|----------------------------------|---------------|
| ctags: creates a tags            | file                             | ctags(CP)     |
| dd: converts and copies a        | file                             | dd(C)         |
| close: closes a                  | file descriptor                  | close(S)      |
| dup2: duplicate an open          | file descriptor                  | dup2(S)       |
| dup, dup2: duplicates an open    | file descriptor                  | dup(S)        |
|                                  | file: determines file type       | file(C)       |
| devices: format of UUCP devices  | file                             | devices(F)    |
| dialers: format of UUCP Dialers  | file                             | dialers(F)    |
| sact: prints current SCCS        | file editing activity            | sact(CP)      |
| crypt: password and              | file encryption functions        | crypt(S)      |
| putpwent: writes a password      | file entry                       | putpwent(S)   |
| utmpname: accesses utmp          | file entry endutent,             | getut(S)      |
| setgrent, endgrent: get group    | file entry /getgrgid, getgrnam,  | getgrent(S)   |
| endpwent: gets password          | file entry /getpwnam, setpwent,  | getpwent(S)   |
| access: determines whether a     | file exists and is accessible    | access(DOS)   |
| filelength: gets the length of a | file                             | fileleng(DOS) |
| fopen: opens a                   | file                             | fopen(DOS)    |
| grep, egrep, fgrep: Searches a   | file for a pattern               | grep(C)       |
| proto: prototype job             | file for at                      | proto(ADM)    |
| mfsys: configuration             | file for filesystem types        | mfsys(F)      |
| open: opens a                    | file for reading or writing      | open(DOS)     |
| open: opens                      | file for reading or writing      | open(S)       |
| ldaopen: open a common object    | file for reading ldopen,         | ldopen(S)     |
| writing sopen: opens a           | file for shared reading and      | sopen(DOS)    |
| uudecode: decode a binary        | file for transmission via mail   | uencode(C)    |
| uencode: encode a binary         | file for transmission via mail   | uencode(C)    |
| ar: archive                      | file format                      | ar(F)         |
| pnch:                            | file format for card images      | pnch(F)       |
| intro: introduction to           | file formats                     | Intro(F)      |
| mkstr: creates an error message  | file from C source               | mkstr(CP)     |
| entries of a common object       | file function /line number       | ldlread(S)    |
| group: format of the group       | file                             | group(M)      |
| grpcheck: checks group           | file                             | grpcheck(C)   |
| files filehdr:                   | file header for common object    | filehdr(F)    |
| implementation-specific/ limits: | file header for                  | limits(F)     |
| constants unistd:                | file header for symbolic         | unistd(F)     |
| file ldfhread: read the          | file header of a common object   | ldfhread(S)   |
| ldohseek: seek to the optional   | file header of a common object/  | ldohseek(S)   |
| changes executable binary        | file headers fixhdr:             | fixhdr(C)     |
| split: Splits a                  | file into pieces                 | split(C)      |
| issue: issue identification      | file                             | issue(F)      |
| ln: makes a link to a            | file                             | ln(C)         |
| mem, kmem: memory image          | file                             | mem(M)        |
| mestbl: create a messages locale | file                             | mestbl(M)     |
| utime: sets                      | file modification time           | utime(DOS)    |
| mtune: tunable parameter         | file                             | mtune(F)      |
| nl: adds line numbers to a       | file                             | nl(C)         |
| null: the null                   | file                             | null(F)       |
| /Finds the slot in the utmp      | file of the current user         | ttyslot(S)    |
| purge(C) purge: the policy       | file of the sanitization utility | purge(F)      |
| rename: renames a                | file or directory                | rename(DOS)   |



|                                  |                                         |                   |
|----------------------------------|-----------------------------------------|-------------------|
| the access permissions of a      | file or directory /Changes . . . .      | chmod(C)          |
| one creat: creates a new         | file or overwrites an existing . . . .  | creat(DOS)        |
| one creat: creates a new         | file or rewrites an existing . . . .    | creat(S)          |
| passwd: the password             | file . . . . .                          | passwd(F)         |
| umask: sets the                  | file permission mask . . . . .          | umask(DOS)        |
| chmod: changes                   | file permissions . . . . .              | chmod(DOS)        |
| fseek: moves a                   | file pointer . . . . .                  | fseek(DOS)        |
| /ftell, rewind: repositions a    | file pointer in a stream . . . . .      | fseek(S)          |
| lseek: changes the position of a | file pointer . . . . .                  | lseek(DOS)        |
| lseek: moves read/write          | file pointer . . . . .                  | lseek(S)          |
| finds the current position of a  | file pointer ftell: . . . . .           | ftell(DOS)        |
| gets the current position of a   | file pointer tell: . . . . .            | tell(DOS)         |
| poll: format of UUCP Poll        | file . . . . .                          | poll(F)           |
| stream fsetpos: sets the         | file position indicator for a . . . . . | fsetpos(DOS)      |
| stream fsetpos: sets the         | file position indicator for a . . . . . | fsetpos(S)        |
| the current value of a stream's  | file position indicator /stores . . . . | fgetpos(DOS)      |
| the current value of a stream's  | file position indicator /stores . . . . | fgetpos(S)        |
| prs: prints an SCCS              | file . . . . .                          | prs(CP)           |
| pwdcheck: checks password        | file . . . . .                          | pwdcheck(C)       |
| read: reads from a               | file . . . . .                          | read(DOS)         |
| read: reads from a               | file . . . . .                          | read(S)           |
| locking: locks or unlocks a      | file region for reading or/ . . . . .   | locking(S)        |
| remove: deletes a                | file . . . . .                          | remove(DOS)       |
| sccsfile: format of an SCCS      | file . . . . .                          | sccsfile(F)       |
| sfsys: local filesystem type     | file . . . . .                          | sfsys(F)          |
| stat: gets                       | file status . . . . .                   | stat(DOS)         |
| stat, fstat: gets                | file status . . . . .                   | stat(S)           |
| mount: mounts a                  | file structure . . . . .                | mount(ADM)        |
| umount: dismounts a              | file structure . . . . .                | umount(ADM)       |
| stune: local tunable parameter   | file . . . . .                          | stune(F)          |
| /symbol name for common object   | file symbol table entry . . . . .       | ldgetname(S)      |
| syms: common object              | file symbol table format . . . . .      | syms(F)           |
| fsdb:                            | file system debugger . . . . .          | fsdb(ADM)         |
| sysfs: get                       | file system type information . . . . .  | sysfs(S)          |
| systems: format of UUCP Systems  | file . . . . .                          | systems(F)        |
| tmpfile: creates a temporary     | file . . . . .                          | tmpfile(S)        |
| freopen: assigns a new           | file to a stream . . . . .              | freopen(DOS)      |
| /converts XENIX-style aliases    | file to MMDF format . . . . .           | mmdfalias(ADM)    |
| /a XENIX-style Micnet routing    | file to MMDF format . . . . .           | mnlist(ADM)       |
| XENIX-style UUCP routing         | file to MMDF format /a . . . . .        | uulist(ADM)       |
| serial/ consoleprint: print      | file to printer attached to a . . . . . | consoleprint(ADM) |
| tsort: Sorts a                   | file topologically . . . . .            | tsort(CP)         |
| the scheduler for the uucp       | file transport program uusched: . . .   | uusched(ADM)      |
| ftw: walks a                     | file tree . . . . .                     | ftw(S)            |
| file: determines                 | file type . . . . .                     | file(C)           |
| unlink: deletes a                | file . . . . .                          | unlink(DOS)       |
| val: validates an SCCS           | file . . . . .                          | val(CP)           |
| write: writes to a               | file . . . . .                          | write(DOS)        |
| write: writes to a               | file . . . . .                          | write(S)          |
| determines accessibility of a    | file access: . . . . .                  | access(S)         |
| format of per-process accounting | file acct: . . . . .                    | acct(F)           |

|                                  |                                         |                   |
|----------------------------------|-----------------------------------------|-------------------|
| header of a common object        | file /an indexed/named section . . .    | ldhread(S)        |
| for and processes a pattern in a | file awk: Searches . . . . .            | awk(C)            |
| changes the owner and group of a | file chown: . . . . .                   | chown(S)          |
| umask: Sets                      | file-creation mode mask . . . . .       | umask(C)          |
| fields of each line of a         | file cut: cuts out selected . . . . .   | cut(C)            |
| a delta (change) to an SCCS      | file delta: makes . . . . .             | delta(CP)         |
| specified in the mdevice         | file /device numbers currently . . .    | majorsinuse(ADM)  |
| of UUCP dial-code abbreviations  | file dialcodes: format . . . . .        | dialcodes(F)      |
| dump selected parts of an object | file dump: . . . . .                    | dump(CP)          |
| execlp, execlvp: executes a      | file /execv, execl, execlve, . . . .    | exec(S)           |
| returns length of target         | file filelength: . . . . .              | filelength(DOS)   |
| directory entries and put in a   | file getdents: read . . . . .           | getdents(S)       |
| object files                     | filehdr: file header for common . . .   | filehdr(F)        |
| alternative login terminals      | file inittab: . . . . .                 | inittab(F)        |
| header of a member of an archive | file ldahread: read the archive . . .   | ldahread(S)       |
| ldaclose: close a common object  | file ldclose, . . . . .                 | ldclose(S)        |
| file header of a common object   | file ldhread: read the . . . . .        | ldhread(S)        |
| symbol table of a common object  | file ldtbseek: seek to the . . . . .    | ldtbseek(S)       |
| file                             | filelength: gets the length of a . . .  | fileleng(DOS)     |
| target file                      | filelength: returns length of . . . .   | filelength(DOS)   |
| entries in a common object       | file linenum: line number . . . . .     | linenum(F)        |
| a new filename to an existing    | file link: links . . . . .              | link(S)           |
| specified in the sdevice         | file /list of vectors currently . . . . | vectorsinuse(ADM) |
| listing from a common object     | file list: produce C source . . . . .   | list(CP)          |
| UUCP uusched limit               | file maxuuscheds: . . . . .             | maxuuscheds(F)    |
| UUCP uuxqt limit                 | file maxuuxqts: . . . . .               | maxuuxqts(F)      |
| the Micnet default commands      | file micnet: . . . . .                  | micnet(F)         |
| or a special or ordinary         | file mknod: makes a directory, . . .    | mknod(S)          |
| driver backend configuration     | file mvdevice: video . . . . .          | mvdevice(F)       |
| ctermid: generates a             | filename for a terminal . . . . .       | ctermid(S)        |
| mktemp: makes a unique           | filename . . . . .                      | mktemp(S)         |
| remove: removes                  | filename . . . . .                      | remove(S)         |
| rename: changes                  | filename . . . . .                      | rename(S)         |
| link: links a new                | filename to an existing file . . . . .  | link(S)           |
| changes the format of a text     | file newform: . . . . .                 | newform(C)        |
| status ferror, feof, clearerr,   | fileno: determines stream . . . . .     | ferror(S)         |
| format of UUCP Permissions       | file permissions: . . . . .             | permissions(F)    |
| table entry of a common object   | file /read an indexed symbol . . . .    | ldtbread(S)       |
| information for a common object  | file reloc: relocation . . . . .        | reloc(F)          |
| remove extra blank lines from a  | file rmb: . . . . .                     | rmb(M)            |
| removes a delta from an SCCS     | file rmdel: . . . . .                   | rmdel(CP)         |
| csplit: Splits                   | files according to context . . . . .    | csplit(C)         |
| rcp: copies                      | files across UNIX systems . . . . .     | rcp(C)            |
| integrity: examine system        | files against authentication/ . . . .   | integrity(ADM)    |
| mv: moves or renames             | files and directories . . . . .         | mv(C)             |
| link, unlink: link and unlink    | files and directories link: . . . . .   | link(ADM)         |
| discretionary attributes of      | files and programs discr: check . . .   | discr(S)          |
| bfs: Scans big                   | files . . . . .                         | bfs(C)            |
| cat: concatenates and displays   | files . . . . .                         | cat(C)            |
| cmp: compares two                | files . . . . .                         | cmp(C)            |
| idmkinit: read                   | files containing specifications . . .   | idmkinit(ADM)     |



|                                   |                                              |                |
|-----------------------------------|----------------------------------------------|----------------|
| copy: copies groups of            | files . . . . .                              | copy(C)        |
| cp: copies                        | files . . . . .                              | cp(C)          |
| diff3: compares three             | files . . . . .                              | diff3(C)       |
| diff: compares two text           | files . . . . .                              | diff(C)        |
| fcntl: controls open              | files . . . . .                              | fcntl(S)       |
| find: finds                       | files . . . . .                              | find(C)        |
| transit queue: MMDf queue         | files for storing mail in . . . . .          | queue(ADM)     |
| translate: translates             | files from one format to another . . . . .   | translate(C)   |
| auditd: read audit collection     | files generated by the audit/ . . . . .      | auditd(ADM)    |
| cleantmp: remove temporary        | files in directories specified . . . . .     | cleantmp(ADM)  |
| hd: displays                      | files in hexadecimal format . . . . .        | hd(C)          |
| od: displays                      | files in octal format . . . . .              | od(C)          |
| /closes and deletes all temporary | files in the current directory . . . . .     | rmtmp(DOS)     |
| mknod: builds special             | files . . . . .                              | mknod(C)       |
| xdummdir: prints the names of     | files on a backup archive . . . . .          | xdummdir(C)    |
| pr: prints                        | files on the standard output . . . . .       | pr(C)          |
| rm, rmdir: removes                | files or directories . . . . .               | rm(C)          |
| paste: merges lines of            | files . . . . .                              | paste(C)       |
| purge: overwrites specified       | files . . . . .                              | purge(C)       |
| sdiff: compares                   | files side-by-side . . . . .                 | sdiff(C)       |
| sort: Sorts and merges            | files . . . . .                              | sort(C)        |
| tar: archives                     | files . . . . .                              | tar(C)         |
| convert: convert archive          | files to common formats . . . . .            | convert(CP)    |
| coffconv: convert 386 COFF        | files to XENIX format . . . . .              | coffconv(M)    |
| bdiff: compares                   | files too large for diff . . . . .           | bdiff(C)       |
| control                           | files, uuinstall: administers UUCP . . . . . | uuinstall(ADM) |
| what: identifies                  | files . . . . .                              | what(C)        |
| and prints process accounting     | files acctcom: Searches for . . . . .        | acctcom(ADM)   |
| merge or add total accounting     | files acctmerg: . . . . .                    | acctmerg(ADM)  |
| creates and administers SCCS      | files admin: . . . . .                       | admin(CP)      |
| compares two versions of an SCCS  | file sccsdiff: . . . . .                     | sccsdiff(CP)   |
| header for a common object        | file scnhdr: section . . . . .               | scnhdr(F)      |
| lines common to two sorted        | files comm: Selects or rejects . . . . .     | comm(C)        |
| format of curses screen image     | file scr_dump: . . . . .                     | scr_dump(F)    |
| local device configuration        | file sdevice: . . . . .                      | sdevice(F)     |
| dosrm, dosrmdir: access DOS       | files dosls, . . . . .                       | dos(C)         |
| section of a common object        | file /seek to an indexed/named . . . . .     | ldsseek(S)     |
| of a section of a common object   | file /seek to relocation entries . . . . .   | ldrseek(S)     |
| file header of a common object    | file /seek to the optional . . . . .         | ldohseek(S)    |
| permissions for a portion of a    | file /sets read and write . . . . .          | locking(DOS)   |
| file header for common object     | files filehdr: . . . . .                     | filehdr(F)     |
| format specification in text      | files fspec: . . . . .                       | fspec(F)       |
| string, format of graphical       | files gps: graphical primitive . . . . .     | gps(F)         |
| parts of executable binary        | files hdr: displays selected . . . . .       | hdr(CP)        |
| to miscellaneous features and     | files intro: introduction . . . . .          | Intro(M)       |
| prints the size of an object      | file size: . . . . .                         | size(CP)       |
| prints the size of an object      | file size: . . . . .                         | size(XNX)      |
| semaphores and record locking on  | files lockf: provide . . . . .               | lockf(S)       |
| format of tty device mapping      | files mapchan: . . . . .                     | mapchan(F)     |
| unpack: compresses and expands    | files pack, pcat, . . . . .                  | pack(C)        |
| access and modification dates of  | files settime: changes the . . . . .         | settime(ADM)   |



|                                      |                                           |               |
|--------------------------------------|-------------------------------------------|---------------|
| fstat: saves                         | file-status information . . . . .         | fstat(DOS)    |
| miscellaneous features and           | files /to machine related . . . . .       | Intro(HW)     |
| top.next: the Micnet topology        | files top, . . . . .                      | top(F)        |
| stop further I/O to an open          | file stopio: . . . . .                    | stopio(S)     |
| printable strings in an object       | file strings: finds the . . . . .         | strings(CP)   |
| checksum and counts blocks in a      | file sum: calculates . . . . .            | sum(C)        |
| gets name list entries from          | files xlist, fxlist: . . . . .            | xlist(S)      |
| format of UUCP Sysfiles              | file sysfiles: . . . . .                  | sysfiles(F)   |
| UNIX system volume fs:               | filesystem - format of . . . . .          | fs(F)         |
| backup: performs incremental         | filesystem backup . . . . .               | backup(ADM)   |
| /AT&T UNIX incremental               | filesystem backup restore . . . . .       | restore(ADM)  |
| interactive, error-checking          | filesystem backup fsave: . . . . .        | fsave(ADM)    |
| XENIX incremental                    | filesystem backup /Performs . . . . .     | xbackup(ADM)  |
| volume                               | filesystem: format of a system . . . . .  | filesystem(F) |
| fstyp: determine                     | filesystem identifier . . . . .           | fstyp(ADM)    |
| directory entry dirent:              | filesystem independent . . . . .          | dirent(F)     |
| fstatfs: get                         | filesystem information . . . . .          | statfs(S)     |
| statfs: get                          | filesystem information . . . . .          | statfs(S)     |
| mkfs: constructs a                   | filesystem . . . . .                      | mkfs(ADM)     |
| mnt: mount a                         | filesystem . . . . .                      | mnt(C)        |
| mount: mounts a                      | filesystem . . . . .                      | mount(S)      |
| quot: Summarizes                     | filesystem ownership . . . . .            | quot(C)       |
| restore, restor: invokes incremental | filesystem restorer . . . . .             | restore(ADM)  |
| XENIX incremental                    | filesystem restorer /Invokes . . . . .    | xrestore(ADM) |
| ustat: gets                          | filesystem statistics . . . . .           | ustat(S)      |
| fsstat: report                       | filesystem status . . . . .               | fsstat(ADM)   |
| mnttab: format of mounted            | filesystem table . . . . .                | mnttab(F)     |
| sfsys: local                         | filesystem type file . . . . .            | sfsys(F)      |
| mfsys: configuration file for        | filesystem types . . . . .                | mfsys(F)      |
| umount: unmounts a                   | filesystem . . . . .                      | umount(S)     |
| prints or changes the name of a      | filesystem fsname: . . . . .              | fsname(ADM)   |
| the Micnet system identification     | file systemid: . . . . .                  | systemid(F)   |
| directory entry dirent:              | file-system-independent . . . . .         | dirent(F)     |
| haltsys, reboot: closes out the      | filesystems and shuts down the/ . . . . . | haltsys(ADM)  |
| /Default information for mounting    | filesystems . . . . .                     | filesys(F)    |
| time dcopy: copy UNIX                | filesystems for optimal access . . . . .  | dcopy(ADM)    |
| fsck: checks and repairs             | filesystems . . . . .                     | fsck(ADM)     |
| labelit: provide labels for          | filesystems . . . . .                     | labelit(ADM)  |
| fsck checklist: list of              | filesystems processed by . . . . .        | checklist(F)  |
| - mount, unmount multiple            | filesystems /umountall . . . . .          | mountall(ADM) |
| literal copy of UNIX                 | filesystem volcopy: make . . . . .        | volcopy(ADM)  |
| delivers the last part of a          | file tail: . . . . .                      | tail(C)       |
| dumps magnetic tape to output        | file tapedump: . . . . .                  | tapedump(C)   |
| format of compiled terminfo          | file "terminfo:" . . . . .                | terminfo(F)   |
| table entry of a common object       | file /the index of a symbol . . . . .     | ldtbindex(S)  |
| creates a name for a temporary       | file tmpnam, tmpnam: . . . . .            | tmpnam(S)     |
| of a section of a common object      | file /to line number entries . . . . .    | ldlseek(S)    |
| and modification times of a          | file touch: updates access . . . . .      | touch(C)      |
| undoes a previous get of an SCCS     | file unget: . . . . .                     | unget(CP)     |
| reports repeated lines in a          | file uniq: . . . . .                      | uniq(C)       |
| uucp directories and permissions     | file uucheck: check the . . . . .         | uucheck(ADM)  |

|                                  |                                          |                    |
|----------------------------------|------------------------------------------|--------------------|
| greek: select terminal           | filter . . . . .                         | greek(C)           |
| col:                             | filters reverse linefeeds . . . . .      | col(C)             |
| tplot: graphics                  | filters . . . . .                        | tplot(ADM)         |
| service lpfilter: administer     | filters used with the LP print . . . .   | lpfilter(ADM)      |
| strstr:                          | finds a string in a string . . . . .     | strstr(DOS)        |
| driver object module routines:   | finds driver entry points in a . . . .   | routines(ADM)      |
| find:                            | finds files . . . . .                    | find(C)            |
| finger:                          | finds information about users . . . .    | finger(C)          |
| logname:                         | finds login name of user . . . . .       | logname(S)         |
| object library lorder:           | finds ordering relation for an . . . .   | lorder(CP)         |
| hashmake, spellin, hashcheck:    | finds spelling errors spell, . . . . .   | spell(C)           |
| file pointer ftell:              | finds the current position of a . . . .  | ftell(DOS)         |
| ttyname, isatty:                 | finds the name of a terminal . . . . .   | ttyname(S)         |
| an object file strings:          | finds the printable strings in . . . .   | strings(CP)        |
| of the current user tytslot:     | finds the slot in the utmp file . . . .  | tytslot(S)         |
| users                            | finger: finds information about . . . .  | finger(C)          |
| dbminit, fetch, store, delete,   | firstkey, nextkey: performs/ . . . .     | dbm(S)             |
| raise: send signal               | sig to execution program . . . . .       | raise(S)           |
| bad track table badtrk: Scans    | fixed disk for flaws and creates . . . . | badtrk(ADM)        |
| binary file headers              | fixhdr: changes executable . . . . .     | fixhdr(C)          |
| badtrk: Scans fixed disk for     | flaws and creates bad track/ . . . .     | badtrk(ADM)        |
| /fpgetsticky, fpsetsticky: IEEE  | floating point environment/ . . . . .    | fpgetround(S)      |
| /isnan: isnand, isnanf: test for | floating point NaN/ . . . . .            | isnan(S)           |
| _control87: gets and sets        | floating-point control word . . . . .    | _control87(DOS)    |
| _fpreset: reinitializes          | floating-point math package . . . . .    | _fpreset(DOS)      |
| frexp, ldexp, modf: Splits       | floating-point number into a/ . . . . .  | frexp(S)           |
| _clear87: gets and clears the    | floating-point status word . . . . .     | _clear87(DOS)      |
| _status87: gets                  | floating-point status word . . . . .     | _status87(DOS)     |
| /fmod: performs absolute value,  | floor, ceiling and remainder/ . . . .    | floor(S)           |
| performs absolute value, floor,/ | floor, fabs, ceil, fmod: . . . . .       | floor(S)           |
| format: format                   | floppy disks . . . . .                   | format(C)          |
| diskcmp: copies or compares      | floppy disks diskcp, . . . . .           | diskcp(C)          |
| fdswap: Swaps default boot       | floppy drive . . . . .                   | fdswap(ADM)        |
| cflow: generates C               | flow graph . . . . .                     | cflow(CP)          |
| buffers                          | flushall: flushes all output . . . . .   | flushall(DOS)      |
| fclose, fflush: closes or        | flushes a stream . . . . .               | fclose(S)          |
| flushall:                        | flushes all output buffers . . . . .     | flushall(DOS)      |
| cPU shutdown:                    | flushes block I/O and halts the . . . .  | shutdown(S)        |
| main memory malloc,              | _fmalloc, _nmalloc: allocate . . . . .   | malloc(DOS)        |
| floor,/ floor, fabs, ceil,       | fmod: performs absolute value, . . . .   | floor(S)           |
| conversions/ fieeeetomsbin,      | fmsbintoieee: perform . . . . .          | fieeeetomsbin(DOS) |
| allocated memory block _msize,   | _fmsize, _nmsize: return size of . . . . | _msize(DOS)        |
| device vidi: Sets the            | font and video mode for a video . . . .  | vidi(C)            |
| stream                           | fopen, freopen, fdopen: opens a . . . .  | fopen(S)           |
|                                  | fopen: opens a file . . . . .            | fopen(DOS)         |
|                                  | fork: creates a new process . . . . .    | fork(S)            |
| ar: archive file                 | format . . . . .                         | ar(F)              |
| backup: incremental dump tape    | format . . . . .                         | backup(F)          |
| nl_ascxtime, nl_cxtime:          | format date and time . . . . .           | nl_cxtime(S)       |
| strftime:                        | format date/time string . . . . .        | strftime(S)        |
| format:                          | format floppy disks . . . . .            | format(C)          |



|                                  |                                                        |                   |
|----------------------------------|--------------------------------------------------------|-------------------|
| pnch: file                       | format for card images . . . . .                       | pnch(F)           |
| 86rel: intel 8086 Relocatable    | format for Object Modules . . . . .                    | 86rel(F)          |
|                                  | format: format floppy disks . . . . .                  | format(C)         |
| od: displays files in octal      | format . . . . .                                       | od(C)             |
|                                  | dir: format of a directory . . . . .                   | dir(F)            |
|                                  | filesystem: format of a system volume . . . . .        | filesystem(F)     |
| newform: changes the             | format of a text file . . . . .                        | newform(C)        |
|                                  | inode: format of an inode . . . . .                    | inode(F)          |
|                                  | scsfile: format of an SCCS file . . . . .              | scsfile(F)        |
| editor output a.out:             | format of assembler and link . . . . .                 | a.out(F)          |
|                                  | file terminfo: format of compiled terminfo . . . . .   | terminfo(F)       |
|                                  | core: format of core image file . . . . .              | core(F)           |
|                                  | cpio: format of cpio archive . . . . .                 | cpio(F)           |
|                                  | file scr_dump: format of curses screen image . . . . . | scr_dump(F)       |
| gps: graphical primitive string, | format of graphical files . . . . .                    | gps(F)            |
| table mnttab:                    | format of mounted filesystem . . . . .                 | mnttab(F)         |
|                                  | file acct: format of per-process accounting . . . . .  | acct(F)           |
|                                  | group: format of the group file . . . . .              | group(M)          |
|                                  | files mapchan: format of tty device mapping . . . . .  | mapchan(F)        |
| volume fs: filesystem -          | format of UNIX system . . . . .                        | fs(F)             |
| volume fs: filesystem            | format of UNIX system . . . . .                        | fs(F)             |
|                                  | devices: format of UUCP devices file . . . . .         | devices(F)        |
| abbreviations file dialcodes:    | format of UUCP dial-code . . . . .                     | dialcodes(F)      |
|                                  | dialers: format of UUCP Dialers file . . . . .         | dialers(F)        |
|                                  | permissions: format of UUCP Permissions file . . . . . | permissions(F)    |
|                                  | poll: format of UUCP Poll file . . . . .               | poll(F)           |
|                                  | sysfiles: format of UUCP Sysfiles file . . . . .       | sysfiles(F)       |
|                                  | systems: format of UUCP Systems file . . . . .         | systems(F)        |
|                                  | sdevice: file format . . . . .                         | sdevice(F)        |
|                                  | sfsys: file format . . . . .                           | sfsys(F)          |
|                                  | files fspec: format specification in text . . . . .    | fspec(F)          |
|                                  | stune: file format . . . . .                           | stune(F)          |
|                                  | tar: archive format . . . . .                          | tar(F)            |
| translates files from one        | format to another translate: . . . . .                 | translate(C)      |
| routing file to MMDF             | format /a XENIX-style Micnet . . . . .                 | mnlist(ADM)       |
| convert 386 COFF files to XENIX  | format coffconv: . . . . .                             | coffconv(M)       |
|                                  | format /converts XENIX-style . . . . .                 | mmdfalias(ADM)    |
| aliases file to MMDF             | format hd: . . . . .                                   | hd(C)             |
| displays files in hexadecimal    | format /perform conversions . . . . .                  | fiectomsbin(DOS)  |
| between IEEE and MS binary       | formats console input . . . . .                        | cscanf(DOS)       |
| cscanf: converts and             | formats input scanf, . . . . .                         | scanf(S)          |
| fscanf, sscanf: converts and     | formats . . . . .                                      | Intro(F)          |
| intro: introduction to file      | formats native language output . . . . .               | nl_printf(S)      |
| /nl_fprintf, nl_sprintf:         | formats of utmp and wtmp . . . . .                     | utmp(F)           |
| entries utmp, wtmp:              | formats output . . . . .                               | cprintf(DOS)      |
| cprintf:                         | formats output . . . . .                               | printf(S)         |
| printf, fprintf, sprintf:        | formats convert: . . . . .                             | convert(CP)       |
| convert archive files to common  | formats /perform conversions . . . . .                 | diecetomsbin(DOS) |
| between MS binary and IEEE       | format syms: . . . . .                                 | syms(F)           |
| common object file symbol table  | formatted native language input . . . . .              | nl_scanf(S)       |
| /nl_fscanf, nl_sscanf: converts  | formatted output of a/ vprintf, . . . . .              | vprintf(S)        |
| vfprintf, vsprintf: prints       |                                                        |                   |



|                                  |                                        |               |
|----------------------------------|----------------------------------------|---------------|
| routing file to MMDF             | format /UUCP . . . . .                 | uulist(ADM)   |
| service lpforms: administer      | forms used with the LP print . . . .   | lpforms(ADM)  |
| ratfor: converts Rational        | FORTRAN into standard FORTRAN          | ratfor(CP)    |
| rational FORTRAN into standard   | FORTRAN ratfor: converts . . . .       | ratfor(CP)    |
| fpgetround, fpsetround,          | fpgetmask, fpsetmask,/ . . . . .       | fpgetround(S) |
| fpgetmask, fpsetmask,/           | fpgetround, fpsetround, . . . . .      | fpgetround(S) |
| floating/ /fpgetmask, fpsetmask, | fpgetsticky, fpsetsticky: IEEE . . .   | fpgetround(S) |
| and segment of an address        | FP_OFF, FP_SEG: get the offset . .     | fp_off(DOS)   |
| and segment                      | fp_off, fp_seg: return offset . . .    | fp_seg(DOS)   |
| floating-point math package      | _fpreset: reinitializes . . . . .      | _fpreset(DOS) |
| output printf,                   | fprintf, sprintf: formats . . . . .    | printf(S)     |
| segment of an address FP_OFF,    | FP_SEG: get the offset and . . . .     | fp_off(DOS)   |
| segment fp_off,                  | fp_seg: return offset and . . . . .    | fp_seg(DOS)   |
| /fpsetround, fpgetmask,          | fpsetmask, fpgetsticky,/ . . . . .     | fpgetround(S) |
| fpsetmask,/ fpgetround,          | fpsetround, fpgetmask, . . . . .       | fpgetround(S) |
| /fpsetmask, fpgetsticky,         | fpsetsticky: IEEE floating point/ .    | fpgetround(S) |
| character to a stream            | fputc, fputchar: write a . . . . .     | fputc(DOS)    |
| word on a/ putc, putchar,        | fputc, putw: puts a character or . .   | putc(S)       |
| stream fputc,                    | fputchar: write a character to a . .   | fputc(DOS)    |
| stream puts,                     | fputs: puts a string on a . . . . .    | puts(S)       |
| binary input and output          | fread, fwrite: performs buffered . .   | fread(S)      |
| stream                           | fread: reads from the input . . . .    | fread(DOS)    |
| memory blocks                    | free, _ffree, _nfree: deallocate . .   | free(DOS)     |
| main memory malloc,              | free, realloc, calloc: allocates . .   | malloc(S)     |
| dynamic memory                   | _freect: counts available . . . . .    | _freect(DOS)  |
| db_alloc, db_free: allocates and | frees physically contiguous/ . . . .   | db_alloc(K)   |
| stream                           | freopen: assigns a new file to a . .   | freopen(DOS)  |
| fopen,                           | freopen, fdopen: opens a stream . .    | fopen(S)      |
| ticks per/ gethz: return the     | frequency of the system clock in .     | gethz(S)      |
| floating-point number into a/    | frexp, ldexp, modf: Splits . . . .     | frexp(S)      |
| genc: create a                   | front-end to the cc command . . . .    | genc(CP)      |
| UNIX system volume               | fs: filesystem - format of . . . . .   | fs(F)         |
| error-checking filesystem/       | fsave: interactive, . . . . .          | fsave(ADM)    |
| formats input scanf,             | fscanf, sscanf: converts and . . .     | scanf(S)      |
| systems                          | fsck: checks and repairs file . . . .  | fsck(ADM)     |
| of filesystems processed by      | fsck checklist: list . . . . .         | checklist(F)  |
| repositions a file pointer in a/ | fsdb: file system debugger . . . . .   | fsdb(ADM)     |
| indicator for a stream           | fseek, ftell, rewind: . . . . .        | fseek(S)      |
| indicator for a stream           | fseek: moves a file pointer . . . .    | fseek(DOS)    |
| name of a filesystem             | fsetpos: sets the file position . . .  | fsetpos(DOS)  |
| text files                       | fsetpos: sets the file position . . .  | fsetpos(S)    |
| semi-automated system backups    | fsname: prints or changes the . . . .  | fsname(ADM)   |
| status                           | fspec: format specification in . . . . | fspec(F)      |
| stat,                            | fsphoto: performs periodic . . . . .   | fsphoto(ADM)  |
| information                      | fsstat: report filesystem . . . . .    | fsstat(ADM)   |
| information                      | fstat: gets file status . . . . .      | stat(S)       |
| identifier                       | fstat: saves file-status . . . . .     | fstat(DOS)    |
| position of a file pointer       | fstatfs: get filesystem . . . . .      | statfs(S)     |
| file pointer in a/ fseek,        | fstyp: determine filesystem . . . .    | fstyp(ADM)    |
|                                  | ftell: finds the current . . . . .     | ftell(DOS)    |
|                                  | ftell, rewind: repositions a . . . .   | fseek(S)      |

|                                  |                                      |               |
|----------------------------------|--------------------------------------|---------------|
|                                  | ftime: gets the current time . . .   | ftime(DOS)    |
| time,                            | ftime: gets time and date . . .      | time(S)       |
| communication package            | ftok: Standard interprocess . . .    | stdipc(S)     |
|                                  | ftw: walks a file tree . . .         | ftw(S)        |
| user data space                  | fubyte: gets a character from . . .  | fubyte(K)     |
| function erf, erfc: error        | function and complementary error .   | erf(S)        |
| gamma: performs log gamma        | function . . . . .                   | gamma(S)      |
| setkey: assigns the              | function keys . . . . .              | setkey(C)     |
| libwindows: windowing terminal   | function library . . . . .           | libwindows(S) |
| matherr: error-handling          | function . . . . .                   | matherr(S)    |
| prof: profile within a           | function . . . . .                   | prof(M)       |
| function and complementary error | function erf, erfc: error . . . . .  | erf(S)        |
| entries of a common object file  | function /manipulate line number .   | ldlread(S)    |
| math: math                       | functions and constants . . . . .    | math(M)       |
| 300: 300, 300s - handle special  | functions of DASI 300/ . . . . .     | 300(C)        |
| terminals hp: handle special     | functions of Hewlett-Packard . . .   | hp(C)         |
| 450/ 450: handle special         | functions of the DASI . . . . .      | 450(C)        |
| sysi86: machine specific         | functions . . . . .                  | sysi86(S)     |
| tcgetattr, tcsetattr: state      | functions . . . . .                  | tcattr(S)     |
| floor, ceiling and remainder     | functions /absolute value, . . . .   | floor(S)      |
| atan2: performs trigonometric    | functions /asin, acos, atan, . . .   | trig(S)       |
| performs UNIX backup             | functions backup: . . . . .          | backup(ADM)   |
| jn, y0, y1, yn: performs Bessel  | functions bessell, j0, j1, . . . .   | bessel(S)     |
| cfsetospeed: baud rate           | functions /cfsetispeed, . . . . .    | cfspeed(S)    |
| password and file encryption     | functions crypt: . . . . .           | crypt(S)      |
| performs screen and cursor       | functions curses: . . . . .          | curses(S)     |
| nextkey: performs database       | functions /delete, firstkey, . . .   | dbm(S)        |
| logarithm, power, square root    | functions /exponential, . . . . .    | exp(S)        |
| to DOS cross development         | functions intro: introduction . . .  | intro(DOS)    |
| cosh, tanh: performs hyperbolic  | functions sinh, . . . . .            | sinh(S)       |
| tcsendbreak: line control        | functions /tcflow, tcflush, . . . .  | tcflow(S)     |
| tcsetpgrp: process group id      | functions tcgetpgrp, . . . . .       | tcpgrp(S)     |
| tgoto, tputs: performs terminal  | functions /tgetflag, tgetstr, . . .  | termcap(S)    |
| stopio: stop                     | further I/O to an open file . . . .  | stopio(S)     |
| from user data space             | fuword: gets one 32-bit word . . .   | fuword(K)     |
| input and output fread,          | fwrite: performs buffered binary .   | fread(S)      |
| stream                           | fwrite: writes to the output . . . . | fwrite(DOS)   |
| manipulate connect accounting/   | fwtmp: fwtmp, wtmpfix: . . . . .     | fwtmp(ADM)    |
| connect accounting/ fwtmp:       | fwtmp, wtmpfix: manipulate . . .     | fwtmp(ADM)    |
| from files xlist,                | fxlist: gets name list entries . . . | xlist(S)      |
| gamma: performs log              | gamma function . . . . .             | gamma(S)      |
| function                         | gamma: performs log gamma . . . .    | gamma(S)      |
| conversions ecvt, fcvt,          | gcvt: performs output . . . . .      | ecvt(S)       |
| cc command                       | genc: create a front-end to the . .  | genc(CP)      |
| adb: invokes a                   | general-purpose debugger . . . . .   | adb(CP)       |
| password randomword:             | generate a pronounceable . . . . .   | randomword(S) |
| user ID diskusg:                 | generate disk accounting data by .   | diskusg(ADM)  |
| and /read audit collection files | generated by the audit subsystem .   | auditd(ADM)   |
| terminal ctermid:                | generates a filename for a . . . .   | ctermid(S)    |
| random:                          | generates a random number . . . .    | random(C)     |
| rand, srand:                     | generates a random number . . . .    | rand(S)       |



|                                 |                                      |               |
|---------------------------------|--------------------------------------|---------------|
| makekey:                        | generates an encryption key . . .    | makekey(M)    |
| abort:                          | generates an IOT fault . . . . .     | abort(S)      |
| cflow:                          | generates C flow graph . . . . .     | cflow(CP)     |
| cross-reference cxref:          | generates C program . . . . .        | cxref(CP)     |
| numbers ncheck:                 | generates names from inode . . .     | ncheck(ADM)   |
| analysis lex:                   | generates programs for lexical . .   | lex(CP)       |
| srand48, seed48, lcong48:       | generates uniformly distributed . .  | drand48(S)    |
| MMDF queue status report        | generator checkque: . . . . .        | checkque(ADM) |
| or set seed for random number   | generator /setseed: obtain . . . .   | seed(S)       |
| value machid: machid, i386      | get processor type truth . . . . .   | machid(C)     |
| block buffer pool geteblk,      | getabl: gets a buffer from the . .   | geteblk(K)    |
| clist buffers                   | getc, getcb, getcbp, getcf: read . . | getc(K)       |
| character or word from a/       | getc, getchar, fgetc, getw: gets . . | getc(S)       |
| buffers getc,                   | getcb, getcbp, getcf: read clist . . | getc(K)       |
| buffers getc, getcb,            | getcbp, getcf: read clist . . . . .  | getc(K)       |
| getc, getcb, getcbp,            | getcf: read clist buffers . . . . .  | getc(K)       |
|                                 | getch: gets a character . . . . .    | getch(DOS)    |
| character or word from a/ getc, | getchar, fgetc, getw: gets . . . .   | getc(S)       |
| input                           | getchar: gets one character of . .   | getchar(K)    |
| character                       | getche: gets and echoes a . . . .    | getche(DOS)   |
| real-time clock                 | getclk: gets string from . . . . .   | getclk(M)     |
| current working directory       | getcwd: get the pathname of . . . .  | getcwd(S)     |
| and put in a file               | getdents: read directory entries . . | getdents(S)   |
| setdvagent, enddvagent,/        | getdvagent, getdvagnam, . . . . .    | getdvagent(S) |
| enddvagent,/ getdvagent,        | getdvagnam, setdvagent, . . . . .    | getdvagent(S) |
| from the block buffer pool      | geteblk, getabl: gets a buffer . .   | geteblk(K)    |
| getuid, geteuid, getgid,        | getegid: gets real user,/ . . . . .  | getuid(S)     |
| environment name                | getenv: gets value for . . . . .     | getenv(S)     |
| real user, effective/ getuid,   | geteuid, getgid, getegid: gets . . . | getuid(S)     |
| effective/ getuid, geteuid,     | getgid, getegid: gets real user, . . | getuid(S)     |
| setgrent, endgrent: get group/  | getgrent, getgrgid, getgrnam, . . .  | getgrent(S)   |
| endgrent: get group/ getgrent,  | getgrgid, getgrnam, setgrent, . . .  | getgrent(S)   |
| get group/ getgrent, getgrgid,  | getgrnam, setgrent, endgrent: . . .  | getgrent(S)   |
| group ID's                      | getgroups: get supplementary . .     | getgroups(S)  |
| the system clock in ticks per/  | gethz: return the frequency of . .   | gethz(S)      |
|                                 | getlogin: gets login name . . . . .  | getlogin(S)   |
|                                 | getluid: get login user ID . . . . . | getluid(S)    |
| stream                          | getmsg: get next message off a . .   | getmsg(S)     |
| argument vector                 | getopt: gets option letter from . .  | getopt(S)     |
|                                 | getopt: parses command options . .   | getopt(C)     |
| options getopt: getopt,         | getoptcv - parse command . . . . .   | getopts(C)    |
| command options getopt:         | getopts, getoptcv - parse . . . . .  | getopts(C)    |
| parse command options           | getopts: getopt, getoptcv - . . . .  | getopts(C)    |
|                                 | getpass: reads a password . . . . .  | getpass(S)    |
| password                        | getpasswd: read or clear a . . . . . | getpasswd(S)  |
| process group, and/ getpid,     | getpgp, getppid: gets process, . .   | getpid(S)     |
| process, process group, and/    | getpid, getpgp, getppid: gets . . .  | getpid(S)     |
| identification number           | getpid: returns a process . . . . .  | getpid(DOS)   |
| group, and/ getpid, getpgp,     | getppid: gets process, process . .   | getpid(S)     |
| setprcmnt, endprcmnt,/          | getprcmnt, getprcmnam, . . . . .     | getprcmnt(S)  |
| endprcmnt,/ getprcmnt,          | getprcmnam, setprcmnt, . . . . .     | getprcmnt(S)  |



|                                  |                                                  |                 |
|----------------------------------|--------------------------------------------------|-----------------|
| setprdfent, endprdfent,/         | getprdfent, getprdfnam, . . . . .                | getprdfent(S)   |
| endprdfent,/ getprdfent,         | getprdfnam, setprdfent, . . . . .                | getprdfent(S)   |
| setprfient, endprfient,/         | getprfient, getprfinam, . . . . .                | getprfient(S)   |
| endprfient,/ getprfient,         | getprfinam, setprfient, . . . . .                | getprfient(S)   |
| associated with this process     | getpriv: get system privileges . . .             | getpriv(S)      |
| getprpwnam, setprpwent,/         | getprpwent, getprpwuid, . . . . .                | getprpwent(S)   |
| getprpwent, getprpwuid,          | getprpwnam, setprpwent,/ . . . .                 | getprpwent(S)   |
| setprpwent,/ getprpwent,         | getprpwuid, getprpwnam, . . . . .                | getprpwent(S)   |
| setprtcent, endprtcent,/         | getprtcent, getprtcnam, . . . . .                | getprtcent(S)   |
| endprtcent,/ getprtcent,         | getprtcnam, setprtcent, . . . . .                | getprtcent(S)   |
| user ID                          | getpw: gets password for a given . .             | getpw(S)        |
| setpwent, endpwent: gets/        | getpwent, getpwuid, getpwnam, . .                | getpwent(S)     |
| gets/ getpwent, getpwuid,        | getpwnam, setpwent, endpwent: . .                | getpwent(S)     |
| endpwent: gets/ getpwent,        | getpwuid, getpwnam, setpwent, . .                | getpwent(S)     |
| buffer pool                      | getblk, getabl: gets a buffer from the block . . | getblk(K)       |
| fgetc, fgetchar:                 | gets a character from a stream . . .             | fgetc(DOS)      |
| space fubyte:                    | gets a character from user data . . .            | fubyte(K)       |
| getch:                           | gets a character . . . . .                       | getch(DOS)      |
| an event queue                   | ev_getdev: gets a list of devices feeding . .    | ev_getdev(S)    |
| shmget:                          | gets a shared memory segment . . .               | shmget(S)       |
| cgets:                           | gets a string . . . . .                          | cgets(DOS)      |
| gets, fgets:                     | gets a string from a stream . . . .              | gets(S)         |
| input gets:                      | gets a string from the standard . .              | gets(CP)        |
| floating-point status/ _clear87: | gets and clears the . . . . .                    | _clear87(DOS)   |
| getche:                          | gets and echoes a character . . . .              | getche(DOS)     |
| control word _control87:         | gets and sets floating-point . . . .             | _control87(DOS) |
| ulimit:                          | gets and sets user limits . . . . .              | ulimit(S)       |
| information dosexterr:           | gets and stores extended error . .               | dosexterr(DOS)  |
| value of a stream's/ fgetpos:    | gets and stores the current . . . .              | fgetpos(DOS)    |
| value of a stream's/ fgetpos:    | gets and stores the current . . . .              | fgetpos(S)      |
| getc, getchar, fgetc, getw:      | gets character or word from a/ . . .             | getc(S)         |
| dosexterr:                       | gets DOS error messages . . . . .                | dosexterr(DOS)  |
| nlist:                           | gets entries from name list . . . .              | nlist(S)        |
| last routine call/ sterror:      | gets error message pointer from . .              | sterror(DOS)    |
| last routine call/ sterror:      | gets error message pointer from . .              | sterror(S)      |
| a stream                         | gets, fgets: gets a string from . . .            | gets(S)         |
| umask: Sets and                  | gets file creation mask . . . . .                | umask(S)        |
| stat:                            | gets file status . . . . .                       | stat(DOS)       |
| stat, fstat:                     | gets file status . . . . .                       | stat(S)         |
| ustat:                           | gets filesystem statistics . . . . .             | ustat(S)        |
| _status87:                       | gets floating-point status word . .              | _status87(DOS)  |
| standard input                   | gets: gets a string from the . . . .             | gets(CP)        |
| getlogin:                        | gets login name . . . . .                        | getlogin(S)     |
| logname:                         | gets login name . . . . .                        | logname(C)      |
| msgget:                          | gets message queue . . . . .                     | msgget(S)       |
| files xlist, fxlist:             | gets name list entries from . . . .              | xlist(S)        |
| system uname:                    | gets name of current UNIX . . . .                | uname(S)        |
| data space fuword:               | gets one 32-bit word from user . .               | fuword(K)       |
| getchar:                         | gets one character of input . . . .              | getchar(K)      |
| vector getopt:                   | gets option letter from argument . .             | getopt(S)       |
| /getpwnam, setpwent, endpwent:   | gets password file entry . . . . .               | getpwent(S)     |

|                                 |                                  |                                   |    |                 |
|---------------------------------|----------------------------------|-----------------------------------|----|-----------------|
| ID                              | getpw:                           | gets password for a given user    | .. | getpw(S)        |
|                                 | times                            | gets process and child process    | .. | times(S)        |
|                                 | getpid, getpgid, getppid:        | gets process, process group, and/ | .. | getpid(S)       |
| real/                           | /geteuid, getgid, getegid:       | gets real user, effective user,   | .. | getuid(S)       |
|                                 | semget:                          | gets set of semaphores            | .. | semget(S)       |
|                                 | getclk:                          | gets string from real-time clock  | .. | getclk(M)       |
| file pointer                    | tell:                            | gets the current position of the  | .. | tell(DOS)       |
|                                 | ftime:                           | gets the current time             | .. | ftime(DOS)      |
|                                 | filelength:                      | gets the length of a file         | .. | filelength(DOS) |
|                                 | cuserid:                         | gets the login name of the user   | .. | cuserid(S)      |
|                                 | tty:                             | gets the terminal's name          | .. | tty(C)          |
|                                 | time, ftime:                     | gets time and date                | .. | time(S)         |
|                                 | getenv:                          | gets value for environment name   | .. | getenv(S)       |
| seed for random number/         | seed:                            | getseed, setseed: obtain or set   | .. | seed(S)         |
| modes, speed, and line/         | ct:                              | getty: Sets terminal type,        | .. | getty(M)        |
|                                 | spawn                            | getty to a remote terminal        | .. | ct(C)           |
| settings used by getty          | "gettydefs:                      | Speed                             | .. | and             |
| and terminal settings used by   | getty                            | "gettydefs:                       | .. | Speed"          |
| security actions for init and   | getty                            | initcond: special                 | .. | initcond(ADM)   |
|                                 | getuid, geteuid, getgid,         | ..                                | .. | getuid(S)       |
| from a/                         | getc, getchar, fgetc,            | getw: gets character or word      | .. | getc(S)         |
| identity: get or check uids or  | gids from program start          | ..                                | .. | identity(S)     |
| of directories                  | ls:                              | gives information about contents  | .. | ls(C)           |
| date and time/                  | ctime, localtime,                | gmtime, asctime, tzset: converts  | .. | ctime(S)        |
|                                 | non-obviousness                  | goodpw: check a password for      | .. | goodpw(ADM)     |
| longjmp: performs a nonlocal    | "goto"                           | setjmp,                           | .. | setjmp(S)       |
| and checks access to a resource | governed by a semaphore          | /Await                            | .. | waitsem(S)      |
| format of graphical files       | gps:                             | graphical primitive string,       | .. | gps(F)          |
| cflow: generates C flow         | graph                            | ..                                | .. | cflow(CP)       |
|                                 | graph: draw a graph              | ..                                | .. | graph(ADM)      |
|                                 | graph                            | ..                                | .. | graph(ADM)      |
|                                 | graph                            | ..                                | .. | sag(ADM)        |
| sag: system activity            | graphical files                  | gps: graphical                    | .. | gps(F)          |
| primitive string, format of     | graphical primitive string,      | ..                                | .. | gps(F)          |
| format of graphical files       | gps:                             | graphical primitive string,       | .. | gps(F)          |
|                                 | tplot:                           | graphics filters                  | .. | tplot(ADM)      |
|                                 | plot:                            | graphics interface                | .. | plot(F)         |
|                                 | plot:                            | graphics interface subroutines    | .. | plot(S)         |
|                                 | greek:                           | select terminal filter            | .. | greek(C)        |
| file for a pattern              | grep, egrep, fgrep:              | Searches a                        | .. | grep(C)         |
| and/                            | /pw_idtoname, gr_nametoid,       | gr_idtoname: map between user     | .. | pw_mapping(S)   |
|                                 | /pw_nametoid, pw_idtoname,       | gr_nametoid, gr_idtoname: map/    | .. | pw_mapping(S)   |
|                                 | /real user, effective user, real | group, and effective group IDs    | .. | getuid(S)       |
| /getppid:                       | gets process, process            | group, and parent process IDs     | .. | getpid(S)       |
| newgrp:                         | logs user into a new             | group                             | .. | newgrp(C)       |
|                                 | passwd: change login,            | group, or dialup shell password   | .. | passwd(C)       |
|                                 | copy: copies                     | groups of files                   | .. | copy(C)         |
| updates, and regenerates        | groups of programs               | /Maintains,                       | .. | make(CP)        |
|                                 | grpcheck: checks group file      | ..                                | .. | grpcheck(C)     |
| signals                         | ssignal,                         | gsignal: implements software      | .. | ssignal(S)      |
|                                 | halloc: allocates a huge array   | ..                                | .. | halloc(DOS)     |
| shutdn:                         | flushes block I/O and            | halts the CPU                     | .. | shutdn(S)       |



|                                  |                                       |                 |
|----------------------------------|---------------------------------------|-----------------|
| panic:                           | halts the system . . . . .            | panic(K)        |
| filesystems and shuts down the/  | haltsys, reboot: closes out the . . . | haltsys(ADM)    |
| DASI 300/ 300: 300, 300s         | handle special functions of . . . .   | 300(C)          |
| DASI/ 300: 300, 300s -           | handle special functions of . . . .   | 300(C)          |
| Hewlett-Packard terminals hp:    | handle special functions of . . . .   | hp(C)           |
| DASI 450 terminal 450:           | handle special functions of the . . . | 450(C)          |
| /strnxfrm, strcoll, strncoll:    | handles collation of strings . . .    | collation(S)    |
| nohup: runs a command immune to  | hangups and quits . . . . .           | nohup(C)        |
| cmchk: reports                   | hard disk block size . . . . .        | cmchk(C)        |
| dparam: displays/changes         | hard disk characteristics . . . . .   | dparam(ADM)     |
| hd: internal                     | hard disk drive . . . . .             | hd(HW)          |
| hcreate, hdestroy: manages       | hash search tables hsearch, . . .     | hsearch(S)      |
| spell, hashmake, spellin,        | hashcheck: finds spelling/ . . . .    | spell(C)        |
| routing/ /builds the MMDF        | hashed database of alias and . . . .  | dbmbuild(ADM)   |
| finds spelling errors spell,     | hashmake, spellin, hashcheck: . . .   | spell(C)        |
| search tables hsearch,           | hcreate, hdestroy: manages hash . .   | hsearch(S)      |
| hexadecimal format               | hd: displays files in . . . . .       | hd(C)           |
|                                  | hd: internal hard disk drive . . .    | hd(HW)          |
| tables hsearch, hcreate,         | hdestroy: manages hash search . . .   | hsearch(S)      |
| executable binary files          | hdr: displays selected parts of . . . | hdr(CP)         |
| scnhdr: section                  | header for a common object file . .   | scnhdr(F)       |
| filehdr: file                    | header for common object files . . .  | filehdr(F)      |
| limits: file                     | header for/ . . . . .                 | limits(F)       |
| unistd: file                     | header for symbolic constants . . .   | unistd(F)       |
| ldfhead: read the file           | header of a common object file . . .  | ldfhead(S)      |
| /seek to the optional file       | header of a common object file . . .  | ldohseek(S)     |
| /read an indexed/named section   | header of a common object file . . .  | ldshread(S)     |
| file ldahread: read the archive  | header of a member of an archive . .  | ldahread(S)     |
| changes executable binary file   | headers fixhdr: . . . . .             | fixhdr(C)       |
| /returns the address of the next | heap entry structure . . . . .        | _fheapwalk(DOS) |
| minimal consistency check on the | heap /_nheapchk: performs a . . .     | _fheapchk(DOS)  |
| user                             | hello: Send a message to another . .  | hello(ADM)      |
| program assert:                  | helps verify validity of . . . . .    | assert(S)       |
| hp: handle special functions of  | Hewlett-Packard terminals . . . . .   | hp(C)           |
| hd: displays files in            | hexadecimal format . . . . .          | hd(C)           |
| block                            | hfree: deallocates a memory . . . .   | hfree(DOS)      |
| layers: protocol used between    | host and windowing terminal/ . . .    | layers(M)       |
| terminal jagent:                 | host control of windowing . . . . .   | jagent(M)       |
| machine: description of          | host machine . . . . .                | machine(HW)     |
| Hewlett-Packard terminals        | hp: handle special functions of . . . | hp(C)           |
| manages hash search tables       | hsearch, hcreate, hdestroy: . . . .   | hsearch(S)      |
| halloc: allocates a              | huge array . . . . .                  | halloc(DOS)     |
| information                      | hwconfig: read the configuration . .  | hwconfig(ADM)   |
| sinh, cosh, tanh: performs       | hyperbolic functions . . . . .        | sinh(S)         |
| euclidean distance               | hypot, cabs: determines . . . . .     | hypot(S)        |
|                                  | i286emul: emulate 80286 . . . . .     | i286emul(C)     |
| value machid: machid,            | i386 - get processor type truth . . . | machid(C)       |
| chgrp: changes group             | ID . . . . .                          | chgrp(C)        |
| chown: changes owner             | ID . . . . .                          | chown(C)        |
| setpgid: set process group       | ID for job control . . . . .          | setpgid(S)      |
| tcsetpgrp: process group         | id functions tcgetpgrp, . . . . .     | tcpgid(S)       |



|                                  |                                  |                   |
|----------------------------------|----------------------------------|-------------------|
| getuid: get login user           | ID                               | getuid(S)         |
| IDs and names                    | id: print user and group         | id(ADM)           |
| and names                        | id: prints user and group IDs    | id(C)             |
| setuid: set login user           | ID                               | setuid(S)         |
| setpgpr: Sets process group      | ID                               | setpgpr(S)        |
| kernel                           | idbuid: build new UNIX system    | idbuid(ADM)       |
| information                      | idcheck: returns selected        | idcheck(ADM)      |
| disk accounting data by user     | ID diskusg: generate             | diskusg(ADM)      |
| issue: issue                     | identification file              | issue(F)          |
| systemid: the Micnet system      | identification file              | systemid(F)       |
| getpid: returns a process        | identification number            | getpid(DOS)       |
| fstyp: determine filesystem      | identifier                       | fstyp(ADM)        |
| devnm:                           | identifies device name           | devnm(C)          |
| what:                            | identifies files                 | what(C)           |
| what:                            | identify SCCS files              | what(CP)          |
| gids from program start          | identity: get or check uids or   | identity(S)       |
| gets password for a given user   | ID getpw:                        | getpw(S)          |
| or get device driver/            | idinstall: add, delete, update,  | idinstall(ADM)    |
| idleout: logs out                | idle users                       | idleout(ADM)      |
|                                  | idleout: logs out idle users     | idleout(ADM)      |
| specifications                   | idmkinit: read files containing  | idmkinit(ADM)     |
| id: print user and group         | IDs and names                    | id(ADM)           |
| id: prints user and group        | IDs and names                    | id(C)             |
| create session and set process   | ID setsid:                       | setsid(S)         |
| get supplementary group          | ID's getgroups:                  | getgroups(S)      |
| group, and parent process        | IDs /Gets process, process       | getpid(S)         |
| between user and group names and | IDs /gr_idtname: map             | pw_mapping(S)     |
|                                  | idspace: investigates free space | idspace(ADM)      |
| real group, and effective group  | IDs /real user, effective user,  | getuid(S)         |
| setgid: Sets user and group      | IDs setuid,                      | setuid(S)         |
| a tunable parameter              | idtune: attempts to set value of | idtune(ADM)       |
| /perform conversions between     | IEEE and MS binary format        | fieeetomsbin(DOS) |
| /fpgetsticky, fpsetsticky:       | IEEE floating point environment/ | fpgetround(S)     |
| between MS binary and            | IEEE formats /conversions        | dieeetomsbin(DOS) |
| core: format of core             | image file                       | core(F)           |
| mem, knem: memory                | image file                       | mem(M)            |
| format of curses screen          | image file scr_dump:             | scr_dump(F)       |
| crash: examine system            | images                           | crash(ADM)        |
| pnch: file format for card       | images                           | pnch(F)           |
| nohup: runs a command            | immune to hangups and quits      | nohup(C)          |
| limits: file header for          | implementation-specific/         | limits(F)         |
| ssignal, gsignal:                | implements software signals      | ssignal(S)        |
| writes a byte to an I/O address  | inb, outb: reads a byte from or  | inb(K)            |
| event input ev_gindev:           | include/exclude devices for      | ev_gindev(S)      |
| backup:                          | incremental backup tape format   | backup(F)         |
| backup: performs                 | incremental filesystem backup    | backup(ADM)       |
| restore: AT&T UNIX               | incremental filesystem backup/   | restore(ADM)      |
| xbackup: performs XENIX          | incremental filesystem backup    | xbackup(ADM)      |
| restore, restor: invokes         | incremental filesystem/          | restore(ADM)      |
| xrestore: invokes XENIX          | incremental filesystem/          | xrestore(ADM)     |
| 32-bit word to a physical I/O/   | ind, outd: reads, writes a       | ind(K)            |

|                                  |                                        |                 |
|----------------------------------|----------------------------------------|-----------------|
| dirent: filesystem               | independent directory entry . . .      | dirent(F)       |
| a common/ ldtbindex: compute the | index of a symbol table entry of . . . | ldtbindex(S)    |
| common object/ ldtbread: read an | indexed symbol table entry of a . . .  | ldtbread(S)     |
| a/ ldshread, ldnsread: read an   | indexed/named section header of . . .  | ldshread(S)     |
| ldsseek, ldnsseek: seek to an    | indexed/named section of a/ . . .      | ldsseek(S)      |
| and teletypes last:              | indicate last logins of users . . .    | last(C)         |
| receipt of an orderly release    | indication /acknowledge . . . . .      | t_rcvrel(NSL)   |
| receive a unit data error        | indication t_rcvuderr: . . . . .       | t_rcvuderr(NSL) |
| fsetpos: sets the file position  | indicator for a stream . . . . .       | fsetpos(DOS)    |
| fsetpos: sets the file position  | indicator for a stream . . . . .       | fsetpos(S)      |
| of a stream's file position      | indicator /the current value . . .     | fgetpos(DOS)    |
| of a stream's file position      | indicator /the current value . . .     | fgetpos(S)      |
| terminfo descriptions            | infocmp: compare or print out . . .    | infocmp(ADM)    |
| /Default backup device           | information . . . . .                  | archive(F)      |
| hwconfig: read the configuration | information . . . . .                  | hwconfig(ADM)   |
| pstat: reports system            | information . . . . .                  | pstat(C)        |
| fstatfs: get filesystem          | information . . . . .                  | statfs(S)       |
| statfs: get filesystem           | information . . . . .                  | statfs(S)       |
| database of alias and routing    | information /hashed . . . . .          | dbmbuild(ADM)   |
| prints lineprinter status        | information lpstat: . . . . .          | lpstat(C)       |
| initialization init,             | inir: process control . . . . .        | init(M)         |
| special security actions for     | init and getty initcond: . . . . .     | initcond(ADM)   |
| initialization                   | init, inir: process control . . . .    | init(M)         |
| actions for init and getty       | initcond: special security . . . .     | initcond(ADM)   |
| init, inir: process control      | initialization . . . . .               | init(M)         |
| printfg: displays driver         | initialization message . . . . .       | printfcg(K)     |
| brc: brc, bcheckrc - system      | initialization procedures . . . . .    | brc(ADM)        |
| allocates contiguous memory at   | initialization memget: . . . . .       | memget(K)       |
| support operation nl_init:       | initializes native language . . . .    | nl_init(S)      |
| t_sndrel:                        | initiate an orderly release . . . .    | t_sndrel(NSL)   |
| process popen, pclose:           | initiates I/O to or from a . . . . .   | popen(S)        |
| terminals file                   | inittab: alternative login . . . . .   | inittab(F)      |
| clri: clears                     | inode . . . . .                        | clri(ADM)       |
| inode: format of an              | inode: format of an inode . . . . .    | inode(F)        |
| ncheck: generates names from     | inode . . . . .                        | inode(F)        |
| inode: format of an              | inode numbers . . . . .                | ncheck(ADM)     |
| inp: returns a byte              | inp: returns a byte . . . . .          | inp(DOS)        |
| fwrite: performs buffered binary | input and output fread, . . . . .      | fread(S)        |
| performs standard buffered       | input and output stdio: . . . . .      | stdio(S)        |
| canon: processes raw             | input data from tty device . . . . .   | canon(K)        |
| getchar: gets one character of   | input . . . . .                        | getchar(K)      |
| fread: reads from the standard   | input stream . . . . .                 | fread(DOS)      |
| pushes character back into       | input stream ungetc: . . . . .         | ungetc(S)       |
| usemouse: maps mouse             | input to keystrokes . . . . .          | usemouse(C)     |
| converts and formats console     | input cscanf: . . . . .                | csanf(DOS)      |
| opens an event queue for         | input ev_open: . . . . .               | ev_open(S)      |
| gets a string from the standard  | input gets: . . . . .                  | gets(CP)        |
| devices for event                | input /include/exclude . . . . .       | ev_gindev(S)    |
| formatted native language        | input /nl_sscanf: converts . . . . .   | nl_scanf(S)     |
| poll: STREAMS                    | input/output multiplexing . . . . .    | poll(S)         |
| sscanf: converts and formats     | input scanf, fscanf, . . . . .         | scanf(S)        |



|                                  |                                            |                 |
|----------------------------------|--------------------------------------------|-----------------|
| uustat: uucp status              | inquiry and job control . . . . .          | uustat(C)       |
| script                           | install: installation shell . . . . .      | install(M)      |
| installpkg:                      | install package . . . . .                  | installpkg(ADM) |
| install:                         | installation shell script . . . . .        | install(M)      |
| xinstall: XENIX                  | installation shell script . . . . .        | xinstall(ADM)   |
| removepkg: remove                | installed package . . . . .                | removepkg(ADM)  |
| displaypkg: display              | installed packages . . . . .               | displaypkg(ADM) |
|                                  | installpkg: install package . . . . .      | installpkg(ADM) |
| creatsem: creates an             | instance of a binary semaphore . . . . .   | creatsem(S)     |
|                                  | int86: executes an interrupt . . . . .     | int86(DOS)      |
|                                  | int86x: executes an interrupt . . . . .    | int86x(DOS)     |
| call                             | intdos: invokes a DOS system . . . . .     | intdos(DOS)     |
| call                             | intdosx: invokes a DOS system . . . . .    | intdosx(DOS)    |
| abs: returns an                  | integer absolute value . . . . .           | abs(S)          |
| /l64a: converts between long     | integer and base 64 ASCII . . . . .        | a64l(S)         |
| sputl, sgetl: accesses long      | integer data in a/ . . . . .               | sputl(S)        |
| the absolute value of a long     | integer labs: returns . . . . .            | labs(DOS)       |
| /l3tol: converts between 3-byte  | integers and long integers . . . . .       | l3tol(S)        |
| div: divides                     | integers . . . . .                         | div(DOS)        |
| div: divides                     | integers . . . . .                         | div(S)          |
| itoa: converts numbers to        | integers . . . . .                         | itoa(DOS)       |
| ldiv: divides long               | integers . . . . .                         | ldiv(DOS)       |
| ldiv: divides long               | integers . . . . .                         | ldiv(S)         |
| ltoa: converts long              | integers to characters . . . . .           | ltoa(DOS)       |
| between 3-byte integers and long | integers /l3tol: converts . . . . .        | l3tol(S)        |
| atoi, atol: converts string to   | integer strtol, . . . . .                  | strtol(S)       |
| against authentication database  | integrity: examine system files . . . . .  | integrity(ADM)  |
| for Object Modules 86rel:        | intel 8086 Relocatable Format . . . . .    | 86rel(F)        |
| filesystem backup fsave:         | interactive, error-checking . . . . .      | fsave(ADM)      |
| program cscope:                  | interactively examine a C . . . . .        | cscope(CP)      |
| audit: audit subsystem           | interface device . . . . .                 | audit(ADM)      |
| activation,/ auditcmd: command   | interface for audit subsystem . . . . .    | auditcmd(ADM)   |
| authtsh: administrator           | interface for authorization/ . . . . .     | authtsh(ADM)    |
| plot: graphics                   | interface . . . . .                        | plot(F)         |
| rtc: real time clock             | interface . . . . .                        | rtc(HW)         |
| scsi: Small computer systems     | interface . . . . .                        | scsi(HW)        |
| plot: graphics                   | interface subroutines . . . . .            | plot(S)         |
| swap: swap administrative        | interface . . . . .                        | swap(ADM)       |
| termio: general terminal         | interface . . . . .                        | termio(M)       |
| termios: pOSIX general terminal  | interface . . . . .                        | termios(M)      |
| /, tty2[a-H], tty2[A-H]:         | interface to serial ports . . . . .        | serial(HW)      |
| tty: Special terminal            | interface . . . . .                        | tty(M)          |
| lp1, lp2: line printer device    | interfaces lp, lp0, . . . . .              | lp(HW)          |
| authentication/ authck: check    | internal consistency of . . . . .          | authck(ADM)     |
| hd:                              | internal hard disk drive . . . . .         | hd(HW)          |
| setlocale: Set or read           | international environment . . . . .        | setlocale(S)    |
| locale: the                      | international locale . . . . .             | locale(M)       |
| spline:                          | interpolates smooth curve . . . . .        | spline(C)       |
| commands tticom:                 | interpret tty driver I/O control . . . . . | tticom(K)       |
| sh: invokes the shell command    | interpreter . . . . .                      | sh(C)           |
| csh: invokes a shell command     | interpreter with C-like syntax . . . . .   | csh(C)          |



- a restricted shell (command interpreter) rsh: invokes . . . . rsh(C)
- ipcs: reports the status of inter-process communication/ . . . ipcs(ADM)
- package ftok: Standard interprocess communication . . . stdipc(S)
- pipe: creates an interprocess pipe . . . . . pipe(S)
- int86: executes an interrupt . . . . . int86(DOS)
- int86x: executes an interrupt . . . . . int86x(DOS)
- spltty, splx: block/permit interrupts /splhi, splni, splpp, . . spl(K)
- sleep: Suspends execution for an interval . . . . . sleep(C)
- sleep: Suspends execution for an interval . . . . . sleep(S)
- suspends execution for a short interval nap: . . . . . nap(S)
- services, library routines and/ intro: introduces system . . . . . Intro(S)
- commands intro: introduces UNIX . . . . . Intro(C)
- development System commands intro: introduces UNIX . . . . . Intro(CP)
- development functions intro: introduction to DOS cross . intro(DOS)
- formats intro: introduction to file . . . . . Intro(F)
- miscellaneous features and/ intro: introduction to . . . . . Intro(M)
- related miscellaneous features/ intro: introduction to machine . . Intro(HW)
- Network Services library intro: introduction to the . . . . intro(NSL)
- references Intro: lists manual page . . . . . Intro(K)
- references intro: lists manual page . . . . . intro(K)
- library routines and/ intro: introduces system services, . . . Intro(S)
- intro: introduces UNIX commands . . . Intro(C)
- system commands intro: introduces UNIX Development . . . Intro(CP)
- development functions intro: introduction to DOS cross . . . . intro(DOS)
- intro: introduction to file formats . . . . . Intro(F)
- miscellaneous features/ intro: introduction to machine related . . Intro(HW)
- features and files intro: introduction to miscellaneous . . Intro(M)
- Services library intro: introduction to the Network . . . intro(NSL)
- idspace: investigates free space . . . . . idspace(ADM)
- bc: invokes a calculator . . . . . bc(C)
- yacc: invokes a compiler-compiler . . . . . yacc(CP)
- bdos: invokes a DOS system call . . . . . bdos(DOS)
- intdos: invokes a DOS system call . . . . . intdos(DOS)
- intdosx: invokes a DOS system call . . . . . intdosx(DOS)
- debugger adb: invokes a general-purpose . . . . . adb(CP)
- m4: invokes a macro processor . . . . . m4(CP)
- calendar: invokes a reminder service . . . . . calendar(C)
- (command interpreter) rsh: invokes a restricted shell . . . . . rsh(C)
- red: invokes a restricted version of . . . . . ed(C)
- display/ vi, view, vedit: invokes a screen-oriented . . . . . vi(C)
- interpreter with C-like/ csh: invokes a shell command . . . . . csh(C)
- ex: invokes a text editor . . . . . ex(C)
- calculator dc: invokes an arbitrary precision . . . . . dc(C)
- restore, restor: invokes incremental filesystem/ . . . restore(ADM)
- sdb: invokes symbolic debugger . . . . . sdb(CP)
- cc: invokes the C compiler . . . . . cc(CP)
- ev\_init: invokes the event manager . . . . . ev\_init(S)
- ld: invokes the link editor . . . . . ld(CP)
- ld: invokes the link editor . . . . . ld(M)
- ld: invokes the link editor . . . . . ld(XNX)
- interpreter sh: invokes the shell command . . . . . sh(C)

|                                  |                                        |                |
|----------------------------------|----------------------------------------|----------------|
| sed:                             | invokes the stream editor . . . .      | sed(C)         |
| ed:                              | invokes the text editor . . . .        | ed(C)          |
| masm:                            | invokes the XENIX assembler . . .      | masm(CP)       |
| incremental file/ xrestore:      | invokes XENIX . . . . .                | xrestore(ADM)  |
| 16-bit word from or to a/        | inw, outw: reads, writes a . . . .     | inw(K)         |
| byte from or writes a byte to an | I/O address inb, outb: reads a . .     | inb(K)         |
| a 32-bit word to a physical      | I/O address /outd: reads, writes . .   | ind(K)         |
| word from or to a physical       | I/O address /writes a 16-bit . . .     | inw(K)         |
| shutdn: flushes block            | i/O and halts the CPU . . . . .        | shutdn(S)      |
| iodone: signals                  | I/O completion . . . . .               | iodone(K)      |
| iowait: wait for                 | I/O completion . . . . .               | iowait(K)      |
| ttiocom: interpret tty driver    | I/O control commands . . . . .         | ttiocom(K)     |
| physio, physck: raw              | I/O for block drivers . . . . .        | physio(K)      |
| select: synchronous              | i/O multiplexing . . . . .             | select(S)      |
| disksort: adds a block           | I/O request to a device's queue . .    | disksort(K)    |
| breaks up programmed             | I/O requests pio_breakup: . . . .      | pio_breakup(K) |
| stopio: stop further             | I/O to an open file . . . . .          | stopio(S)      |
| popen, pclose: initiates         | i/O to or from a process . . . . .     | popen(S)       |
| devices                          | ioctl: controls character . . . . .    | ioctl(S)       |
|                                  | iodone: signals I/O completion . .     | iodone(K)      |
| abort: generates an              | iOT fault . . . . .                    | abort(S)       |
|                                  | iowait: wait for I/O completion . .    | iowait(K)      |
| semaphore set or shared memory   | ipcrm: removes a message queue, .      | ipcrm(ADM)     |
| inter-process communication/     | ipcs: reports the status of . . . .    | ipcs(ADM)      |
| /islower, isdigit, isxdigit,     | isalnum, isspace, ispunct,/ . . . .    | ctype(S)       |
| isdigit, isxdigit,/ ctype,       | isalpha, isupper, islower, . . . .     | ctype(S)       |
| /isprint, isgraph, isctrl,       | isascii, tolower, toupper,/ . . . .    | ctype(S)       |
| device                           | isatty: checks for a character . . .   | isatty(DOS)    |
| terminal ttyname,                | isatty: finds the name of a . . . .    | ttyname(S)     |
| /ispunct, isprint, isgraph,      | isctrl, isascii, tolower,/ . . . .     | ctype(S)       |
| /isalpha, isupper, islower,      | isdigit, isxdigit, isalnum,/ . . . .   | ctype(S)       |
| /isspace, ispunct, isprint,      | isgraph, isctrl, isascii,/ . . . .     | ctype(S)       |
| ctype, isalpha, isupper,         | islower, isdigit, isxdigit,/ . . . .   | ctype(S)       |
| state                            | ismpx: return windowing terminal .     | ismpx(C)       |
| floating point NaN/              | isnan: isnand, isnanf: test for . . .  | isnan(S)       |
| floating point NaN/ isnan:       | isnanand, isnanf: test for . . . .     | isnan(S)       |
| NaN/ isnan: isnand,              | isnanf: test for floating point . . .  | isnan(S)       |
| /isalnum, isspace, ispunct,      | isprint, isgraph, isctrl,/ . . . .     | ctype(S)       |
| /isxdigit, isalnum, isspace,     | ispunct, isprint, isgraph,/ . . . .    | ctype(S)       |
| /isdigit, isxdigit, isalnum,     | isspace, ispunct, isprint,/ . . . .    | ctype(S)       |
| issue:                           | issue identification file . . . . .    | issue(F)       |
|                                  | issue: issue identification file . . . | issue(F)       |
| isxdigit,/ ctype, isalpha,       | isupper, islower, isdigit, . . . .     | ctype(S)       |
| /isupper, islower, isdigit,      | isxdigit, isalnum, isspace,/ . . . .   | ctype(S)       |
|                                  | item: CRT item routines . . . . .      | item(S)        |
| item: CRT                        | item routines . . . . .                | item(S)        |
| news: print news                 | items . . . . .                        | news(C)        |
| integers                         | itoa: converts numbers to . . . .      | itoa(DOS)      |
| Bessel functions bessel,         | j0, j1, jn, y0, y1, yn: performs . .   | bessel(S)      |
| Bessel functions bessel, j0,     | j1, jn, y0, y1, yn: performs . . . .   | bessel(S)      |
| windowing terminal               | jagent: host control of . . . . .      | jagent(M)      |



|                                  |                                                 |                                 |
|----------------------------------|-------------------------------------------------|---------------------------------|
| functions                        | bessel, j0, j1, jn, y0, y1, yn: performs Bessel | bessel(S)                       |
|                                  | join: joins two relations                       | join(C)                         |
|                                  | join: joins two relations                       | join(C)                         |
|                                  | terminal                                        | jterm: reset layer of windowing |
| sigsetjmp, siglongjmp: non-local | jumps                                           | sigsetjmp(S)                    |
|                                  | jwin: print size of layer                       | jwin(C)                         |
| keystroke                        | kbhit: checks the console for a                 | kbhit(DOS)                      |
| test keyboard support            | kbmode: Set keyboard mode or                    | kbmode(ADM)                     |
|                                  | error: kernel error output device               | error(M)                        |
| idbuild: build new UNIX system   | kernel                                          | idbuild(ADM)                    |
| /selfailure, selwakeup:          | kernel routines supporting/                     | select(K)                       |
| bcopy: copies bytes in           | kernel space                                    | bcopy(K)                        |
| copies bytes between user and    | kernel space copyin, copyout:                   | copyin(K)                       |
| builds a new UNIX system         | kernel link_unix:                               | link_unix(ADM)                  |
| between user space and the       | kernel /passc: passes character                 | cpass(K)                        |
| makekey: generates an encryption | key                                             | makekey(M)                      |
| keyboard: the PC                 | keyboard                                        | keyboard(HW)                    |
| support kbmode: Set              | keyboard mode or test keyboard                  | kbmode(ADM)                     |
| set keyboard mode or test        | keyboard support kbmode:                        | kbmode(ADM)                     |
|                                  | keyboard: the PC keyboard                       | keyboard(HW)                    |
| setkey: assigns the function     | keys                                            | setkey(C)                       |
| kbhit: checks the console for a  | keystroke                                       | kbhit(DOS)                      |
| usemouse: maps mouse input to    | keystrokes                                      | usemouse(C)                     |
|                                  | kill all active processes                       | killall(ADM)                    |
| process or a group of/           | kill: Sends a signal to a                       | kill(S)                         |
|                                  | kill: terminates a process                      | kill(C)                         |
| processes                        | killall: kill all active                        | killall(ADM)                    |
| mem,                             | kmem: memory image file                         | mem(M)                          |
| physical addresses ptok,         | ktop: converts virtual and                      | ptok(K)                         |
| contents of directory            | l: lists information about                      | l(C)                            |
| 3-byte integers and long/        | l3tol, ltol3: converts between                  | l3tol(S)                        |
| integer and base 64/ a64l,       | l64a: converts between long                     | a64l(S)                         |
| systems                          | labelit: provide labels for file                | labelit(ADM)                    |
| labelit: provide                 | labels for filesystems                          | labelit(ADM)                    |
|                                  | labs: converts to absolute value                | labs(S)                         |
| of a long integer                | labs: returns the absolute value                | labs(DOS)                       |
| nl_langinfo:                     | language information                            | nl_langinfo(S)                  |
| converts formatted native        | language input /nl_sscanf:                      | nl_sscanf(S)                    |
| nl_sprintf: formats native       | language output /nl_fprintf,                    | nl_printf(S)                    |
| cpp: the C                       | language preprocessor                           | cpp(CP)                         |
| nl_strncmp: compare native       | language strings nl_strcmp,                     | nl_strcmp(S)                    |
| nl_init: initializes native      | language support operation                      | nl_init(S)                      |
| lint: checks C                   | language usage and syntax                       | lint(CP)                        |
| /chargefee, ckpacct, dodisk,     | lastlogin, monacct, nulladm,/                   | acctsh(ADM)                     |
| jwin: print size of              | layer                                           | jwin(C)                         |
| shl: Shell                       | layer manager                                   | shl(C)                          |
| terminals                        | layers: layer multiplexer for windowing         | layers(C)                       |
| jterm: reset                     | layer of windowing terminal                     | jterm(C)                        |
| login entry to show current      | layer relogin: rename                           | relogin(ADM)                    |
| windowing terminals              | layers: layer multiplexer for                   | layers(C)                       |
| host and windowing terminal/     | layers: protocol used between                   | layers(M)                       |



|                                  |                                       |                 |
|----------------------------------|---------------------------------------|-----------------|
| columns                          | lc: lists directory contents in . . . | lc(C)           |
| distributed srand48, seed48,     | lcong48: generates uniformly . . .    | drand48(S)      |
|                                  | ld: invokes the link editor . . .     | ld(CP)          |
|                                  | ld: invokes the link editor . . .     | ld(M)           |
|                                  | ld: invokes the link editor . . .     | ld(XNX)         |
| file ldclose,                    | ldclose: close a common object . .    | ldclose(S)      |
| header of a member of an/        | ldahread: read the archive . . .      | ldahread(S)     |
| file for reading ldopen,         | ldlopen: open a common object . .     | ldopen(S)       |
| common object file               | ldclose, ldaclose: close a . . .      | ldclose(S)      |
| floating-point number/ frexp,    | ldexp, modf: Splits . . . . .         | frexp(S)        |
| routines                         | ldfcn: common object file access .    | ldfcn(F)        |
| of a common object file          | ldhread: read the file header . . .   | ldhread(S)      |
| for common object file symbol/   | ldgetname: retrieve symbol name . .   | ldgetname(S)    |
|                                  | ldiv: divides long integers . . . .   | ldiv(DOS)       |
|                                  | ldiv: divides long integers . . . .   | ldiv(S)         |
| line number entries of/ ldread,  | ldlinit, ldlimem: manipulate . . .    | ldlread(S)      |
| entries of a/ ldread, ldlimem,   | ldlimem: manipulate line number .     | ldlread(S)      |
| manipulate line number entries/  | ldlread, ldlimem, ldlimem: . . . .    | ldlread(S)      |
| number entries of a section of/  | ldlseek, ldnlseek: seek to line . .   | ldlseek(S)      |
| entries of a section/ ldseek,    | ldnlseek: seek to line number . . .   | ldlseek(S)      |
| entries of a section/ ldrseek,   | ldnrseek: seek to relocation . . .    | ldrseek(S)      |
| indexed/named/ ldshread,         | ldnshread: read an . . . . .          | ldshread(S)     |
| indexed/named section/ ldsseek,  | ldnsseek: seek to an . . . . .        | ldsseek(S)      |
| file header of a common object/  | ldohseek: seek to the optional . .    | ldohseek(S)     |
| object file for reading          | ldopen, ldaopen: open a common .      | ldopen(S)       |
| relocation entries of a section/ | ldrseek, ldnrseek: seek to . . . .    | ldrseek(S)      |
| indexed/named section header/    | ldshread, ldnsread: read an . . .     | ldshread(S)     |
| indexed/named section of a/      | ldsseek, ldnsseek: seek to an . .     | ldsseek(S)      |
| a symbol table entry of a/       | ldtbind: compute the index of . .     | ldtbind(S)      |
| table entry of a common object/  | ldtbread: read an indexed symbol .    | ldtbread(S)     |
| table of a common object file    | ldtbseek: seek to the symbol . . .    | ldtbseek(S)     |
| filelength: gets the             | length of a file . . . . .            | fileleng(DOS)   |
| strlen: returns the              | length of a string . . . . .          | strlen(DOS)     |
| filelength: returns              | length of target file . . . . .       | filelength(DOS) |
| determine minimum password       | length: passlen: . . . . .            | passlen(S)      |
| getopt: gets option              | letter from argument vector . . .     | getopt(S)       |
| banner: prints large             | letters . . . . .                     | banner(C)       |
| lexical analysis                 | lex: generates programs for . . .     | lex(CP)         |
| lex: generates programs for      | lexical analysis . . . . .            | lex(CP)         |
| linear array search              | lfind, lsearch: performs a . . . .    | lfind(DOS)      |
| and update lsearch,              | lfind: performs linear search . . .   | lsearch(S)      |
| ar: maintains archives and       | libraries . . . . .                   | ar(CP)          |
| ar: maintains archives and       | libraries . . . . .                   | ar(XNX)         |
| chkshlib: compare shared         | libraries tool . . . . .              | chkshlib(CP)    |
| converts archives to random      | libraries: ranlib: . . . . .          | ranlib(CP)      |
| mkshlib: create a shared         | library . . . . .                     | mkshlib(CP)     |
| /Introduces system services,     | library routines and error/ . . .     | Intro(S)        |
| field: FIELD                     | library routines . . . . .            | field(S)        |
| fieldtype: FIELDTYPE             | library routines . . . . .            | fieldtype(S)    |
| form: FORM                       | library routines . . . . .            | form(S)         |
| t_alloc: allocate a              | library structure . . . . .           | t_alloc(NSL)    |

|                                  |                                            |                  |
|----------------------------------|--------------------------------------------|------------------|
| t_free: free a                   | library structure . . . . .                | t_free(NSL)      |
| t_sync: synchronize transport    | library . . . . .                          | t_sync(NSL)      |
| to the Network Services          | library intro: introduction . . . . .      | intro(NSL)       |
| windowing terminal function      | library libwindows: . . . . .              | libwindows(S)    |
| ordering relation for an object  | library lorder: finds . . . . .            | lorder(CP)       |
| function library                 | libwindows: windowing terminal . . . . .   | libwindows(S)    |
| maxuuscheds: UUCP uusched        | limit file . . . . .                       | maxuuscheds(F)   |
| maxuuxqts: UUCP uuxqt            | limit file . . . . .                       | maxuuxqts(F)     |
| implementation-specific/         | limits: file header for . . . . .          | limits(F)        |
| ulimit: gets and sets user       | limits . . . . .                           | ulimit(S)        |
| line: reads one                  | line . . . . .                             | line(C)          |
| lfind, lsearch: performs a       | linear array search . . . . .              | lfind(DOS)       |
| lsearch, lfind: performs         | linear search and update . . . . .         | lsearch(S)       |
| profile data lprof: display      | line-by-line execution count . . . . .     | lprof(CP)        |
| col: filters reverse             | linefeeds . . . . .                        | col(C)           |
| a common object file             | linenum: line number entries in . . . . .  | linenum(F)       |
| lpshut, lpmove: Starts/stops the | lineprinter request lpsched, . . . . .     | lpsched(ADM)     |
| lpadmin: configures the          | lineprinter spooling system . . . . .      | lpadmin(ADM)     |
| lpstat: prints                   | lineprinter status information . . . . .   | lpstat(C)        |
| cancel: Send/cancel requests to  | lineprinter lp . . . . .                   | lp(C)            |
| adds, reconfigures and maintains | lineprinters lpinit: . . . . .             | lpinit(ADM)      |
| files comm: Selects or rejects   | lines common to two sorted . . . . .       | comm(C)          |
| rmb: remove extra blank          | lines from a file . . . . .                | rmb(M)           |
| uniq: reports repeated           | lines in a file . . . . .                  | uniq(C)          |
| head: prints the first few       | lines of a stream . . . . .                | head(C)          |
| paste: merges                    | lines of files . . . . .                   | paste(C)         |
| wc: counts                       | lines, words and characters . . . . .      | wc(C)            |
| directories link: link, unlink:  | link and unlink files and . . . . .        | link(ADM)        |
| ld: invokes the                  | link editor . . . . .                      | ld(CP)           |
| ld: invokes the                  | link editor . . . . .                      | ld(M)            |
| ld: invokes the                  | link editor . . . . .                      | ld(XNX)          |
| a.out: format of assembler and   | link editor output . . . . .               | a.out(F)         |
| unlink files and directories     | link: link, unlink: link and . . . . .     | link(ADM)        |
| existing file                    | link: links a new filename to an . . . . . | link(S)          |
| ln: makes a                      | link to a file . . . . .                   | ln(C)            |
| files and directories link:      | link, unlink: link and unlink . . . . .    | link(ADM)        |
| dosld: XENIX to MS-DOS cross     | linker . . . . .                           | dosld(CP)        |
| os2ld: OS/2 cross                | linker . . . . .                           | os2ld(CP)        |
| existing file link:              | links a new filename to an . . . . .       | link(S)          |
| UNIX system kernel               | link_unix: builds a new . . . . .          | link_unix(ADM)   |
| and syntax                       | lint: checks C language usage . . . . .    | lint(CP)         |
| xlist, fxlist: gets name         | list entries from files . . . . .          | xlist(S)         |
| MMDF                             | list: list processor channel for . . . . . | list(ADM)        |
| nlist: gets entries from name    | list . . . . .                             | nlist(S)         |
| nm: prints name                  | list . . . . .                             | nm(CP)           |
| nm: prints name                  | list . . . . .                             | nm(XNX)          |
| queue ev_getdev: gets a          | list of devices feeding an event . . . . . | ev_getdev(S)     |
| by fsck checklist:               | list of filesystems processed . . . . .    | checklist(F)     |
| majorsinuse: displays the        | list of major device numbers/ . . . . .    | majorsinuse(ADM) |
| terminals:                       | list of supported terminals . . . . .      | terminals(M)     |
| swconfig: produces a             | list of the software/ . . . . .            | swconfig(C)      |



|                                  |                                  |                   |
|----------------------------------|----------------------------------|-------------------|
| vectorsinuse: displays the       | list of vectors currently/       | vectorsinuse(ADM) |
| list:                            | list processor channel for MMDF  | list(ADM)         |
| from a common object file        | list: produce C source listing   | list(CP)          |
| setgroups: set group access      | list                             | setgroups(S)      |
| varargs: variable argument       | list                             | varargs(S)        |
| t_listen:                        | listen for a connect request     | t_listen(NSL)     |
| nlsadmin: network                | listener service administration  | nlsadmin(ADM)     |
| cref: makes a cross-reference    | listing                          | cref(CP)          |
| file list: produce C source      | listing from a common object     | list(CP)          |
| of a varargs argument            | list /Prints formatted output    | vprintf(S)        |
| columns lc:                      | lists directory contents in      | lc(C)             |
| of directory l:                  | lists information about contents | l(C)              |
| Intro:                           | lists manual page references     | Intro(K)          |
| intro:                           | lists manual page references     | intro(K)          |
| who:                             | lists who is on the system       | who(C)            |
| filesystem volcopy: make         | literal copy of UNIX             | volcopy(ADM)      |
| /execv, execve, execvp, execvpe: | ln: makes a link to a file       | ln(C)             |
| mestbl: create a messages        | load and execute a process       | exec(DOS)         |
| locale: the international        | locale file                      | mestbl(M)         |
| chrtbl: create a ctype           | locale                           | locale(M)         |
| coltbl: create a collation       | locale table                     | chrtbl(M)         |
| monthbl: create a currency       | locale table                     | coltbl(M)         |
| numtbl: create a numeric         | locale table                     | monthbl(M)        |
| locale                           | locale table                     | numtbl(M)         |
| network mmdf: routes mail        | locale: the international        | locale(M)         |
| tzset: converts date and/ ctime, | locally and over any supported   | mmdf(ADM)         |
| end, etext, edata: last          | localtime, gmtime, asctime,      | ctime(S)          |
| bzero: sets memory               | locations in program             | end(S)            |
| memory                           | locations to 0 (zero)            | bzero(K)          |
| memory plock:                    | lock: locks a process in primary | lock(S)           |
| database dblock:                 | lock: locks a user's terminal    | lock(C)           |
| record locking on files          | lock process, text, or data in   | plock(S)          |
| region for reading or writing    | lock the entire Authentication   | dblock(S)         |
| provide semaphores and record    | lockf: provide semaphores and    | lockf(S)          |
| permissions for a portion of a/  | locking: locks or unlocks a file | locking(S)        |
| memory lock:                     | locking on files lockf:          | lockf(S)          |
| lock:                            | locking: sets read and write     | locking(DOS)      |
| for reading or/ locking:         | locks a process in primary       | lock(S)           |
| gamma: performs                  | locks a user's terminal          | lock(C)           |
| exponential, logarithm,/ exp,    | locks or unlocks a file region   | locking(S)        |
| logarithm,/ exp, log, pow, sqrt, | log gamma function               | gamma(S)          |
| /log10: performs exponential,    | log, pow, sqrt, log10: performs  | exp(S)            |
| logs: MMDF                       | log10: performs exponential,     | exp(S)            |
| strclean: STREAMS error          | logarithm, power, square root/   | exp(S)            |
| strerr: STREAMS error            | logfiles                         | logs(F)           |
| layer relogin: rename            | logger cleanup program           | strclean(ADM)     |
| password passwd: change          | logger daemon                    | strerr(ADM)       |
| getlogin: gets                   | login entry to show current      | relogin(ADM)      |
| logname: gets                    | login, group, or dialup shell    | passwd(C)         |
|                                  | login name                       | getlogin(S)       |
|                                  | login name                       | logname(C)        |



|                                  |                                        |               |
|----------------------------------|----------------------------------------|---------------|
| cuserid: gets the                | login name of the user . . . . .       | cuserid(S)    |
| logname: finds                   | login name of user . . . . .           | logname(S)    |
| passwd: changes                  | login password . . . . .               | passwd(C)     |
| terminal:                        | login terminal . . . . .               | terminal(HW)  |
| inittab: alternative             | login terminals file . . . . .         | inittab(F)    |
| sets up an environment at        | login time profile: . . . . .          | profile(M)    |
| getluid: get                     | login user ID . . . . .                | getluid(S)    |
| setluid: set                     | login user ID . . . . .                | setluid(S)    |
| last: indicate last              | logins of users and teletypes . . .    | last(C)       |
| user                             | logname: finds login name of . . .     | logname(S)    |
|                                  | logname: gets login name . . . .       | logname(C)    |
|                                  | logs: MMDF logfiles . . . . .          | logs(F)       |
| idleout:                         | logs out idle users . . . . .          | idleout(ADM)  |
| newgrp:                          | logs user into a new group . . . .     | newgrp(C)     |
| call with error                  | longjmp: ends current system . . .     | longjmp(K)    |
| "goto" setjmp,                   | longjmp: performs a nonlocal . . .     | setjmp(S)     |
| for an object library            | lorder: finds ordering relation . . .  | lorder(CP)    |
| uppercase strupr: converts       | lowercase characters to . . . . .      | strupr(DOS)   |
| converts uppercase characters to | lowercase strlwr: . . . . .            | strlwr(DOS)   |
| requests to lineprinter          | lp, cancel: Send/cancel . . . . .      | lp(C)         |
| device interfaces                | lp, lp0, lp1, lp2: line printer . . .  | lp(HW)        |
| utility lpsh: menu driven        | lp print service administration . .    | lpsh(ADM)     |
| administer filters used with the | IP print service lpfilter: . . . . .   | lpfilter(ADM) |
| administer forms used with the   | IP print service lpforms: . . . . .    | lpforms(ADM)  |
| device interfaces lp,            | lp0, lp1, lp2: line printer . . . . .  | lp(HW)        |
| interfaces lp, lp0,              | lp1, lp2: line printer device . . . .  | lp(HW)        |
| interfaces lp, lp0, lp1,         | lp2: line printer device . . . . .     | lp(HW)        |
| lineprinter spooling system      | lpadmin: configures the . . . . .      | lpadmin(ADM)  |
| used with the LP print service   | lpfilter: administer filters . . . . . | lpfilter(ADM) |
| with the LP print service        | lpforms: administer forms used . .     | lpforms(ADM)  |
| maintains lineprinters           | lpinit: adds, reconfigures and . . .   | lpinit(ADM)   |
| lineprinter/ lp sched, lpshut,   | lpmove: Starts/stops the . . . . .     | lp sched(ADM) |
| attached to the user's terminal  | lprint: print to a printer . . . . .   | lprint(C)     |
| execution count profile data     | lprof: display line-by-line . . . . .  | lprof(CP)     |
| starts/stops the lineprinter/    | lp sched, lpshut, lpmove: . . . . .    | lp sched(ADM) |
| service administration utility   | lpsh: menu driven lp print . . . . .   | lpsh(ADM)     |
| lineprinter request lp sched,    | lpshut, lpmove: Starts/stops the . .   | lp sched(ADM) |
| status information               | lpstat: prints lineprinter . . . . .   | lpstat(C)     |
| priorities                       | lpusers: set printing queue . . . .    | lpusers(ADM)  |
| contents of directories          | ls: gives information about . . . . .  | ls(C)         |
| search and update                | lsearch, lfind: performs linear . .    | lsearch(S)    |
| search lfind,                    | lsearch: performs a linear array . .   | lfind(DOS)    |
| file pointer                     | lseek: changes the position of a . .   | lseek(DOS)    |
| pointer                          | lseek: moves read/write file . . . .   | lseek(S)      |
| characters                       | ltoa: converts long integers to . . .  | ltoa(DOS)     |
| integers and long/ l3tol,        | l3tol3: converts between 3-byte . . .  | l3tol(S)      |
|                                  | m4: invokes a macro processor . . .    | m4(CP)        |
| type truth value machid:         | machid, i386 - get processor . . . .   | machid(C)     |
| processor type truth value       | machid: machid, i386 - get . . . . .   | machid(C)     |
| machine                          | machine: description of host . . . .   | machine(HW)   |
| port /verify                     | machine is suitable for security . .   | check_data(S) |

|                                  |                                      |                  |
|----------------------------------|--------------------------------------|------------------|
| machine: description of host     | machine . . . . .                    | machine(HW)      |
| features/ intro: introduction to | machine related miscellaneous . . .  | Intro(HW)        |
| sysi86:                          | machine specific functions . . . .   | sysi86(S)        |
| values:                          | machine-dependent values . . . .     | values(M)        |
| accesses long integer data in a  | machine-independent /sgetl: . . .    | sputl(S)         |
| m4: invokes a                    | macro processor . . . . .            | m4(CP)           |
| program tape:                    | magnetic tape maintenance . . .      | tape(C)          |
| tapedump: dumps                  | magnetic tape to output file . . .   | tapedump(C)      |
| deliver: MMDF                    | mail delivery process . . . . .      | deliver(ADM)     |
| submit: MMDF                     | mail enqueueer . . . . .             | submit(ADM)      |
| MMDF queue files for storing     | mail in transit queue: . . . . .     | queue(ADM)       |
| supported network mmdf: routes   | mail locally and over any . . . .    | mmdf(ADM)        |
| rmail: submit remote             | mail received via UUCP . . . .       | rmail(ADM)       |
| of mail                          | mail: Sends, reads or disposes . .   | mail(C)          |
| but not/ checkmail: checks for   | mail which has been submitted . .    | checkmail(C)     |
| daemon.mn: micnet                | mailer daemon . . . . .              | daemon.mn(M)     |
| sends, reads or disposes of      | mail mail: . . . . .                 | mail(C)          |
| binary file for transmission via | mail uudecode: decode a . . . .      | uencode(C)       |
| binary file for transmission via | mail uuencode: decode a . . . .      | uuencode(C)      |
| _fmalloc, _nmalloc: allocate     | main memory malloc, . . . . .        | malloc(DOS)      |
| free, realloc, calloc: allocates | main memory malloc, . . . . .        | malloc(S)        |
| fdisk:                           | maintain disk partitions . . . . .   | fdisk(ADM)       |
| libraries ar:                    | maintains archives and . . . . .     | ar(CP)           |
| ar:                              | maintains archives and libraries . . | ar(XNX)          |
| lpinit: adds, reconfigures and   | maintains lineprinters . . . . .     | lpinit(ADM)      |
| regenerates groups of/ make:     | maintains, updates, and . . . . .    | make(CP)         |
| sys tty: System                  | maintenance device . . . . .         | sys tty(M)       |
| tape: magnetic tape              | maintenance program . . . . .        | tape(C)          |
| major, new device number, or/    | major, makedev, minor: returns . .   | major(K)         |
| major, makedev, minor: returns   | major, new device number, or/ . .    | major(K)         |
| of major device numbers/         | majorsinuse: displays the list . . . | majorsinuse(ADM) |
| new device number, or/ major,    | makedev, minor: returns major, . .   | major(K)         |
| key                              | makekey: generates an encryption .   | makekey(M)       |
| cref:                            | makes a cross-reference listing . .  | cref(CP)         |
| execseg:                         | makes a data region executable . .   | execseg(S)       |
| SCCS file delta:                 | makes a delta (change) to an . . .   | delta(CP)        |
| mkdir:                           | makes a directory . . . . .          | mkdir(C)         |
| or ordinary file mknod:          | makes a directory, or a special . .  | mknod(S)         |
| ln:                              | makes a link to a file . . . . .     | ln(C)            |
| mktemp:                          | makes a unique filename . . . . .    | mktemp(S)        |
| another user su:                 | makes the user a super-user or . .   | su(C)            |
| allocate main memory             | malloc, _fmalloc, _nmalloc: . . .    | malloc(DOS)      |
| allocates main memory            | malloc, free, realloc, calloc: . . . | malloc(S)        |
| endpoint t_optmgmt:              | manage options for a transport . .   | t_optmgmt(NSL)   |
| ev_init: invokes the event       | manager . . . . .                    | ev_init(S)       |
| shl: Shell layer                 | manager . . . . .                    | shl(C)           |
| tsearch, tfind, tdelete, twalk:  | manages binary search trees . . .    | tsearch(S)       |
| hsearch, hcreate, hdestroy:      | manages hash search tables . . .     | hsearch(S)       |
| /endprcment, putprcmmam:         | manipulate command control/ . . .    | getprcment(S)    |
| records fwtmtp: fwtmtp, wtmtpfx: | manipulate connect accounting . .    | fwtmtp(ADM)      |
| /endprdfent, putprdfnam:         | manipulate default control/ . . .    | getprdfent(S)    |



|                                     |                                  |                 |
|-------------------------------------|----------------------------------|-----------------|
| /putdvagname, copydvagent:          | manipulate device assignment/    | getdvagent(S)   |
| entry /endprfient, putprfinam:      | manipulate file control database | getprfient(S)   |
| of a/ ldhread, ldinit, lditem:      | manipulate line number entries   | ldhread(S)      |
| /endprpwent, putprpwnam:            | manipulate protected password/   | getprpwnent(S)  |
| sigset:                             | manipulate signal sets           | sigset(S)       |
| /endprtcent, putprtcnam:            | manipulate terminal control/     | getprtcent(S)   |
| comment section mcs:                | manipulate the object file       | mcs(CP)         |
| Subsystems/ subsystems:             | manipulation routines for        | subsystems(S)   |
| /floating-point number into a       | mantissa and an exponent         | frexp(S)        |
| Intro: lists                        | manual page references           | Intro(K)        |
| intro: lists                        | manual page references           | intro(K)        |
| and/ /gr_nametoid, gr_idtoname:     | map between user and group names | pw_mapping(S)   |
| ascii:                              | map of the ASCII character set   | ascii(M)        |
| mapping                             | mapchan: configure tty device    | mapchan(M)      |
| mapping files                       | mapchan: format of tty device    | mapchan(F)      |
| convkey: configure monitor/         | mapkey, mapscrm, mapstr,         | mapkey(M)       |
| emdupmap: duplicates channel        | mapping                          | emdupmap(K)     |
| mapchan: format of tty device       | mapping files                    | mapchan(F)      |
| mapchan: configure tty device       | mapping                          | mapchan(M)      |
| emunmap: disables                   | mapping on a channel             | emunmap(K)      |
| configure monitor screen            | mapping /mapstr, convkey:        | mapkey(M)       |
| /allocates temporary memory or      | maps a device into memory        | sptalloc(K)     |
| usemouse:                           | maps mouse input to keystrokes   | usemouse(C)     |
| configure monitor/ mapkey, mapscrm, | mapscrm, mapstr, convkey:        | mapkey(M)       |
| monitor screen/ mapkey, mapscrm,    | mapstr, convkey: configure       | mapkey(M)       |
| ev_setemask: sets event             | mask                             | ev_setemask(S)  |
| ev_setemask: Sets event             | mask                             | ev_stemask(S)   |
| umask: Sets file-creation mode      | mask                             | umask(C)        |
| umask: sets the file permission     | mask                             | umask(DOS)      |
| return the current event            | mask ev_getemask:                | ev_getemask(S)  |
| return the current event            | mask ev_getemask:                | ev_gtemask(S)   |
| sets and gets file creation         | mask umask:                      | umask(S)        |
| assembler                           | masm: invokes the XENIX          | masm(CP)        |
| regular expression compile and      | match routines regexp:           | regexp(S)       |
| math:                               | math functions and constants     | math(M)         |
| constants                           | math: math functions and         | math(M)         |
| reinitializes floating-point        | math package _fpreset:           | _fpreset(DOS)   |
| function                            | matherr: error-handling          | matherr(S)      |
| limit file                          | maxuuscheds: UUCP uuxched        | maxuuscheds(F)  |
| limit file                          | maxuuxqts: UUCP uuxqt            | maxuuxqts(F)    |
| comment section                     | mcs: manipulate the object file  | mcs(CP)         |
| currently specified in the          | mdevice file /device numbers     | majorinuse(ADM) |
|                                     | mem, kmem: memory image file     | mem(M)          |
| available memory                    | _memavl: returns size of         | _memavl(DOS)    |
| /read the archive header of a       | member of an archive file        | ldhread(S)      |
| memory at initialization            | memget: allocates contiguous     | memget(K)       |
| byte-by-byte                        | memicmp: compares buffers        | memicmp(DOS)    |
| between objects                     | memmove: copies characters       | memmove(DOS)    |
| between objects                     | memmove: copies characters       | memmove(S)      |
| memget: allocates contiguous        | memory at initialization         | memget(K)       |
| hfree: deallocates a                | memory block                     | hfree(DOS)      |



|                                  |                                    |               |
|----------------------------------|------------------------------------|---------------|
| size of a previously allocated   | memory block /changes the . . .    | _expand(DOS)  |
| return size of allocated         | memory block /_fmsize, _nmsize:    | _msize(DOS)   |
| free, _ffree, _nfree: deallocate | memory blocks . . . . .            | free(DOS)     |
| mem, kmem:                       | memory image file . . . . .        | mem(M)        |
| bzero: sets                      | memory locations to 0 (zero) . . . | bzero(K)      |
| lock: locks a process in primary | memory . . . . .                   | lock(S)       |
| shmctl: controls shared          | memory operations . . . . .        | shmctl(S)     |
| shmop: performs shared           | memory operations . . . . .        | shmop(S)      |
| sptalloc: allocates temporary    | memory or maps a device into/ . .  | sptalloc(K)   |
| between bytes and clicks         | (memory pages) /ctob: converts .   | btoc(K)       |
| sptalloc sptfree: releases       | memory previously allocated with . | sptfree(K)    |
| vasunbind: virtual address space | memory routines /vasmapped, . .    | vas(K)        |
| shmget: gets a shared            | memory segment . . . . .           | shmget(S)     |
| reports virtual                  | memory statistics vmstat: . . . .  | vmstat(C)     |
| /transfers data from physical    | memory to a user address . . . .   | db_read(K)    |
| memory or maps a device into     | memory /allocates temporary . .    | sptalloc(K)   |
| and frees physically contiguous  | memory /db_free: allocates . . .   | db_alloc(K)   |
| a user address to contiguous     | memory db_write: transfers from .  | db_write(K)   |
| counts available dynamic         | memory _freet: . . . . .           | _freet(DOS)   |
| _mmalloc: allocate main          | memory malloc, _fmalloc, . . . .   | malloc(DOS)   |
| realloc, calloc: allocates main  | memory malloc, free, . . . . .     | malloc(S)     |
| returns size of available        | memory _memavl: . . . . .          | _memavl(DOS)  |
| lock process, text, or data in   | memory plock: . . . . .            | plock(S)      |
| queue, semaphore set or shared   | memory /Removes a message . .      | ipcrm(ADM)    |
|                                  | menu: CRT menu routines . . . .    | menu(S)       |
| administration/ atcronsh:        | menu driven at and cron . . . .    | atcronsh(ADM) |
| utility auditsh:                 | menu driven audit administration . | auditsh(ADM)  |
| administration/ backupsh:        | menu driven backup . . . . .       | backupsh(ADM) |
| administration utility lpsh:     | menu driven lp print service . .   | lpsh(ADM)     |
| administration/ sysadmsh:        | menu driven system . . . . .       | sysadmsh(ADM) |
| menu: CRT                        | menu routines . . . . .            | menu(S)       |
| files acctmerg:                  | merge or add total accounting . .  | acctmerg(ADM) |
| sort: Sorts and                  | merges files . . . . .             | sort(C)       |
| paste:                           | merges lines of files . . . . .    | paste(C)      |
| sent to a terminal               | mesg: permits or denies messages . | mesg(C)       |
| msgctl: provides                 | message control operations . . .   | msgctl(S)     |
| mkstr: creates an error          | message file from C source . . .   | mkstr(CP)     |
| getmsg: get next                 | message off a stream . . . . .     | getmsg(S)     |
| putmsg: send a                   | message on a stream . . . . .      | putmsg(S)     |
| deverr: prints a device error    | message on the console . . . . .   | deverr(K)     |
| printf: print a                  | message on the console . . . . .   | printf(K)     |
| msgop:                           | message operations . . . . .       | msgop(S)      |
| cmn_err: displays                | message or panics the system . . . | cmn_err(K)    |
| routine/ strerror: gets error    | message pointer from last . . . .  | strerror(DOS) |
| routine/ strerror: gets error    | message pointer from last . . . .  | strerror(S)   |
| msgget: gets                     | message queue . . . . .            | msgget(S)     |
| shared memory ipcrm: removes a   | message queue, semaphore set or .  | ipcrm(ADM)    |
| t_error: produce error           | message . . . . .                  | t_error(NSL)  |
| hello: Send a                    | message to another user . . . . .  | hello(ADM)    |
| displays driver initialization   | message printfg: . . . . .         | printfg(K)    |
| console messages                 | messages: description of system .  | messages(M)   |

|                                  |                                            |                |
|----------------------------------|--------------------------------------------|----------------|
| dosexterr: gets DOS error        | messages . . . . .                         | dosexter(DOS)  |
| mestbl: create a                 | messages locale file . . . . .             | mestbl(M)      |
| mesg: permits or denies          | messages sent to a terminal . . . . .      | mesg(C)        |
| description of system console    | messages messages: . . . . .               | messages(M)    |
| prints STREAMS trace             | messages strace: . . . . .                 | strace(ADM)    |
| errno: Sends system error        | messages /sys_nerr, . . . . .              | perror(S)      |
| file                             | mestbl: create a messages locale           | mestbl(M)      |
| filesystem types                 | mfsys: configuration file for . . . . .    | mfsys(F)       |
| micnet: the                      | micnet default commands file . . . . .     | micnet(F)      |
| daemon.mn:                       | micnet mailer daemon . . . . .             | daemon.mn(M)   |
| mnlist: converts a XENIX-style   | micnet routing file to/ . . . . .          | mnlist(ADM)    |
| file systemid: the               | micnet system identification . . . . .     | systemid(F)    |
| commands file                    | micnet: the Micnet default . . . . .       | micnet(F)      |
| top, top.next: the               | micnet topology files . . . . .            | top(F)         |
| _fheapchk, _nheapchk: performs a | minimal consistency check on the/          | _fheapchk(DOS) |
| passlen: determine               | minimum password length . . . . .          | passlen(S)     |
| major, new device number, or     | minor device number /returns . . . . .     | major(K)       |
| number, or/ major, makedev,      | minor: returns major, new device           | major(K)       |
| /- overview of accounting and    | miscellaneous accounting/ . . . . .        | acct(ADM)      |
| /Introduction to machine related | miscellaneous features and/ . . . . .      | Intro(HW)      |
| files intro: introduction to     | miscellaneous features and . . . . .       | Intro(M)       |
|                                  | mkdir: creates a new directory . . . . .   | mkdir(DOS)     |
|                                  | mkdir: make a directory . . . . .          | mkdir(S)       |
|                                  | mkdir: makes a directory . . . . .         | mkdir(C)       |
| special file                     | mkfifo: make a FIFO . . . . .              | mkfifo(S)      |
|                                  | mkfs: constructs a filesystem . . . . .    | mkfs(ADM)      |
|                                  | mknod: builds special files . . . . .      | mknod(C)       |
| special or ordinary file         | mknod: makes a directory, or a             | mknod(S)       |
|                                  | mkshlib: create a shared library . . . . . | mkshlib(CP)    |
| file from C source               | mkstr: creates an error message . . . . .  | mkstr(CP)      |
|                                  | mktemp: makes a unique filename . . . . .  | mktemp(S)      |
| calendar time                    | mktime: converts local time to . . . . .   | mktime(S)      |
| /Micnet routing file to          | MMDF format . . . . .                      | mnlist(ADM)    |
| /UUCP routing file to            | MMDF format . . . . .                      | uulist(ADM)    |
| XENIX-style aliases file to      | MMDF format /converts . . . . .            | mmdfalias(ADM) |
| alias and/ dbmbuild: builds the  | MMDF hashed database of . . . . .          | dbmbuild(ADM)  |
| list: list processor channel for | MMDF . . . . .                             | list(ADM)      |
| logs:                            | MMDF logfiles . . . . .                    | logs(F)        |
| deliver:                         | MMDF mail delivery process . . . . .       | deliver(ADM)   |
| submit:                          | MMDF mail enqueueuer . . . . .             | submit(ADM)    |
| tables:                          | MMDF Name Tables . . . . .                 | tables(F)      |
| mail in transit queue:           | MMDF queue files for storing . . . . .     | queue(ADM)     |
| generator checkque:              | MMDF queue status report . . . . .         | checkque(ADM)  |
| over any supported network       | mmndf: routes mail locally and . . . . .   | mmndf(ADM)     |
| aliases file to MMDF/            | mmdfalias: converts XENIX-style . . . . .  | mmdfalias(ADM) |
| micnet routing file to/          | mnlist: converts a XENIX-style . . . . .   | mnlist(ADM)    |
|                                  | mnt: mount a filesystem . . . . .          | mnt(C)         |
| system table                     | mnttab: format of mounted file . . . . .   | mnttab(F)      |
| vidi: Sets the font and video    | mode for a video device . . . . .          | vidi(C)        |
| umask: Sets file-creation        | mode mask . . . . .                        | umask(C)       |
| chmod: changes                   | mode of a file . . . . .                   | chmod(S)       |



|                                  |                                        |                   |
|----------------------------------|----------------------------------------|-------------------|
| kbmode: Set keyboard             | mode or test keyboard support . .      | kbmode(ADM)       |
| setmode: Sets translation        | mode . . . . .                         | setmode(DOS)      |
| sulogin: access single-user      | mode . . . . .                         | sulogin(ADM)      |
| dial: dials a                    | modem . . . . .                        | dial(ADM)         |
| uuchat: dials a                  | modem . . . . .                        | dial(ADM)         |
| getty: Sets terminal type,       | modes, speed, and line/ . . . .        | getty(M)          |
| uugetty: set terminal type,      | modes, speed, and line/ . . . .        | uugetty(ADM)      |
| tset: Sets terminal              | modes . . . . .                        | tset(C)           |
| number into a/ frexp, ldexp,     | modf: Splits floating-point . . . .    | frexp(S)          |
| settime: changes the access and  | modification dates of files . . . .    | settime(ADM)      |
| utime: sets file                 | modification time . . . . .            | utime(DOS)        |
| touch: updates access and        | modification times of a file . . . .   | touch(C)          |
| utime: sets file access and      | modification times . . . . .           | utime(S)          |
| /produces a list of the software | modifications to the system . . . .    | swconfig(C)       |
| entry points in a driver object  | module routines: finds driver . . . .  | routines(ADM)     |
| relocatable Format for Object    | modules 86rel: intel 8086 . . . .      | 86rel(F)          |
| /ckpacct, dodisk, lastlogin,     | monacct, nulladm, prctmp/ . . . .      | acctsh(ADM)       |
| profile                          | monitor: prepares execution . . . .    | monitor(S)        |
| /mapstr, convkey: configure      | monitor screen mapping . . . . .       | mapkey(M)         |
| tty[01-n], color,                | monochrome, ega, screen: . . . .       | screen(HW)        |
| table                            | montbl: create a currency locale . .   | montbl(M)         |
| mnt:                             | mount a filesystem . . . . .           | mnt(C)            |
|                                  | mount: mounts a file structure . . . . | mount(ADM)        |
|                                  | mount: mounts a filesystem . . . . .   | mount(S)          |
| mountall: mountall, umountall -  | mount, unmount multiple file/ . . .    | mountall(ADM)     |
| mountall: mountall, umountall    | mount, unmount multiple file/ . . .    | mountall(ADM)     |
| mount, unmount multiple file/    | mountall: mountall, umountall - . .    | mountall(ADM)     |
| unmount multiple file/ mountall: | mountall, umountall - mount, . . .     | mountall(ADM)     |
| mnttab: format of                | mounted filesystem table . . . . .     | mnttab(F)         |
| /Default information for         | mounting filesystems . . . . .         | filesystem(F)     |
| mount:                           | mounts a file structure . . . . .      | mount(ADM)        |
| mount:                           | mounts a filesystem . . . . .          | mount(S)          |
| usemouse: maps                   | mouse input to keystrokes . . . . .    | usemouse(C)       |
| mouse: System                    | mouse . . . . .                        | mouse(HW)         |
|                                  | mouse: System mouse . . . . .          | mouse(HW)         |
| specific address                 | movedata: copies bytes from a . . .    | movedata(DOS)     |
| mvdir:                           | moves a directory . . . . .            | mvdir(C)          |
| fseek:                           | moves a file pointer . . . . .         | fseek(DOS)        |
| directories mv:                  | moves or renames files and . . . .     | mv(C)             |
| lseek:                           | moves read/write file pointer . . .    | lseek(S)          |
| /perform conversions between     | MS binary and IEEE formats . . . .     | diecetomsbin(DOS) |
| conversions between IEEE and     | MS binary format /perform . . . .      | fiecetomsbin(DOS) |
| utility                          | mscreen: Serial multiscreens . . . .   | mscreen(M)        |
| dosld: XENIX to                  | mS-DOS cross linker . . . . .          | dosld(CP)         |
| operations                       | msgctl: provides message control . .   | msgctl(S)         |
|                                  | msgget: gets message queue . . . .     | msgget(S)         |
|                                  | msgop: message operations . . . . .    | msgop(S)          |
| size of allocated memory block   | _msize, _fmsize, _nmsize: return . .   | _msize(DOS)       |
|                                  | mtune: tunable parameter file . . . .  | mtune(F)          |
| umountall - mount, unmount       | multiple filesystems /mountall, . .    | mountall(ADM)     |
| used by xt(HW)/ xtproto:         | multiplexed channels protocol . . .    | xtproto(M)        |



# Permuted Index

## Commands, System Calls, Library Routines and File Formats

This permuted index is derived from the "Name" description lines found on each reference manual page. Each *index* line shows the title of the entry to which the line refers, followed by the reference manual section letter where the page is found.

To use the *permuted index* search the middle column for a key word or phrase. The right hand column contains the name and section letter of the manual page that documents the key word or phrase. The left column contains additional useful information about the command. Commands or routines are also listed in the context of the *index* line, followed by a colon (:). This denotes the "beginning" of the sentence. Notice that in many cases, the lines wrap, starting in the middle column and ending in the left column. A slash (/) indicates that the description line is truncated.

|                                |                                     |                |
|--------------------------------|-------------------------------------|----------------|
| panel: PANEL library           | routines . . . . .                  | panel(S)       |
| make a FIFO special            | file mkfifo: . . . . .              | mkfifo(S)      |
| inw, outw: reads, writes a     | 16-bit word from or to a/ . . . .   | inw(K)         |
| what: identify SCCS            | files . . . . .                     | what(CP)       |
| panel: PANEL                   | library routines . . . . .          | panel(S)       |
| functions of DASI 300/         | 300: 300, 300s - handle special . . | 300(C)         |
| functions of DASI/ 300:        | 300, 300s - handle special . . . .  | 300(C)         |
| functions of DASI              | 300 and 300s terminals /special .   | 300(C)         |
| of DASI 300/ 300: 300,         | 300s - handle special functions . . | 300(C)         |
| of DASI 300 and                | 300s terminals /functions . . . .   | 300(C)         |
| fuword: gets one               | 32-bit word from user data space .  | fuword(K)      |
| suword: stores a               | 32-bit word in user data space . .  | suword(K)      |
| ind, outd: reads, writes a     | 32-bit word to a physical I/O/ . .  | ind(K)         |
| coffconv: convert              | 386 COFF files to XENIX format . .  | coffconv(M)    |
| l3tol, ltol3: converts between | 3-byte integers and long/ . . . .   | l3tol(S)       |
| TEKTRONIX 4014 terminal        | 4014: paginator for the . . . . .   | 4014(C)        |
| paginator for the TEKTRONIX    | 4014 terminal 4014: . . . . .       | 4014(C)        |
| the DASI 450 terminal          | 450: handle special functions of .  | 450(C)         |
| functions of the DASI          | 450 terminal /handle special . . .  | 450(C)         |
| accepts a number of            | 512-byte blocks . . . . .           | login(M)       |
| sizes DMA request into         | 512-byte blocks dma_breakup: . .    | dma_breakup(K) |
| /object downloader for the     | 5620 DMD terminal . . . . .         | wtinit(ADM)    |
| between long integer and base  | 64 ASCII a64l, l64a: converts . . . | a64l(S)        |
| i286emul: emulate              | 80286 . . . . .                     | i286emul(C)    |
| x286emul: emulate XENIX        | 80286 . . . . .                     | x286emul(C)    |
| object Modules 86rel: intel    | 8086 Relocatable Format for . . .   | 86rel(F)       |
| asx: XENIX                     | 8086/186/286/386 Assembler . .      | asx(CP)        |
| format for Object Modules      | 86rel: intel 8086 Relocatable . .   | 86rel(F)       |
| long integer and base 64 ASCII | a64l, l64a: converts between . . .  | a64l(S)        |
| format of UUCP dial-code       | abbreviations file dialcodes: . . . | dialcodes(F)   |

|                                  |                                      |               |
|----------------------------------|--------------------------------------|---------------|
|                                  | abort: generates an IOT fault . . .  | abort(S)      |
|                                  | abort: terminates a process . . .    | abort(DOS)    |
| value                            | abs: returns an integer absolute . . | abs(S)        |
| abs: returns an integer          | absolute value . . . . .             | abs(S)        |
| and/ /fabs, ceil, fmod: performs | absolute value, floor, ceiling . .   | floor(S)      |
| labs: converts to                | absolute value . . . . .             | labs(S)       |
| number cabs: calculates the      | absolute value of a complex . . .    | cabs(DOS)     |
| integer labs: returns the        | absolute value of a long . . . .     | labs(DOS)     |
| t_accept:                        | accept a connect request . . . .     | t_accept(NSL) |
| if password is cryptic           | acceptable_password: determine .     | accept_pw(S)  |
| blocks                           | accepts a number of 512-byte . .     | login(M)      |
| files settime: changes the       | access and modification dates of .   | settime(ADM)  |
| a file touch: updates            | access and modification times of .   | touch(C)      |
| utime: sets file                 | access and modification times . .    | utime(S)      |
| record/ /audit_close: open and   | access audit session data on a . .   | audit(S)      |
| of a file                        | access: determines accessibility .   | access(S)     |
| file exists and is accessible    | access: determines whether a . .     | access(DOS)   |
| dosls, dosrm, dosrmdir:          | access DOS files . . . . .           | dos(C)        |
| setgroups: set group             | access list . . . . .                | setgroups(S)  |
| directory chmod: changes the     | access permissions of a file or . .  | chmod(C)      |
| ldfcn: common object file        | access routines . . . . .            | ldfcn(F)      |
| sulogin:                         | access single-user mode . . . . .    | sulogin(ADM)  |
| filesystems for optimal          | access time /copy UNIX . . . .       | dcopy(ADM)    |
| sdenter, sdleave: synchronizes   | access to a shared data segment .    | sdenter(S)    |
| sputl, sgetl:                    | accesses long integer data in a/ .   | sputl(S)      |
| endutent, utmpname:              | accesses utmp file entry . . . .     | getut(S)      |
| access: determines               | accessibility of a file . . . . .    | access(S)     |
| whether a file exists and is     | accessible access: determines . .    | access(DOS)   |
| synchronizes shared data         | access sdgetv, sdwaitv: . . . .      | sdgetv(S)     |
| csplit: splits files             | according to context . . . . .       | csplit(C)     |
| accton: turns on                 | accounting . . . . .                 | accton(ADM)   |
| /accton, acctwtmp - overview of  | accounting and miscellaneous/ . .    | acct(ADM)     |
| of accounting and miscellaneous  | accounting commands /- overview .    | acct(ADM)     |
| diskusg: generate disk           | accounting data by user ID . . . .   | diskusg(ADM)  |
| acct: format of per-process      | accounting file . . . . .            | acct(F)       |
| acctmrg: merge or add total      | accounting files . . . . .           | acctmrg(ADM)  |
| searches for and prints process  | accounting files acctcom: . . . .    | acctcom(ADM)  |
| command summary from per-process | accounting records acctcms: . . .    | acctcms(ADM)  |
| wtmpfix: manipulate connect      | accounting records /fwtmp, . . . .   | fwtmp(ADM)    |
| runacct: run daily               | accounting . . . . .                 | runacct(ADM)  |
| enables or disables process      | accounting acct: . . . . .           | acct(S)       |
| acctprc1, acctprc2 - process     | accounting acctprc: . . . . .        | acctprc(ADM)  |
| turnacct - shell procedures for  | accounting /shutacct, startup, . .   | acctsh(ADM)   |
| accton, acctwtmp - overview of/  | acct: acctdisk, acctdusg, . . . .    | acct(ADM)     |
| process accounting               | acct: enables or disables . . . .    | acct(S)       |
| accounting file                  | acct: format of per-process . . . .  | acct(F)       |
| per-process accounting records   | acctcms: command summary from . .    | acctcms(ADM)  |
| process accounting files         | acctcom: searches for and prints .   | acctcom(ADM)  |
| acctwtmp - overview of/ acct:    | acctdisk, acctdusg, accton, . . . .  | acct(ADM)     |
| overview of/ acct: acctdisk,     | acctdusg, accton, acctwtmp - . . .   | acct(ADM)     |
| accounting files                 | acctmrg: merge or add total . . .    | acctmrg(ADM)  |



|                                  |                                  |                 |
|----------------------------------|----------------------------------|-----------------|
| windowng terminals xt:           | multiplexed tty driver for AT&T  | xt(HW)          |
| terminals layers: layer          | multiplexer for windowing        | layers(C)       |
| select: synchronous I/O          | multiplexing                     | select(S)       |
| STREAMS input/output             | multiplexing poll:               | poll(S)         |
| mscreen: Serial                  | multiscreens utility             | mscreen(M)      |
| rc2: run commands performed for  | multiuser environment            | rc2(ADM)        |
| directories                      | mv: moves or renames files and   | mv(C)           |
| configuration file               | mvdevice: video driver backend   | mvdevice(F)     |
|                                  | mvdir: moves a directory         | mvdir(C)        |
| devnm: identifies device         | name                             | devnm(C)        |
| getlogin: gets login             | name                             | getlogin(S)     |
| logname: gets login              | name                             | logname(C)      |
| pwd: prints working directory    | name                             | pwd(C)          |
| tty: gets the terminal's         | name                             | tty(C)          |
| gets value for environment       | name getenv:                     | getenv(S)       |
| map between user and group       | names and IDs /gr_idtoname:      | pw_mapping(S)   |
| ncheck: generates                | names from inode numbers         | ncheck(ADM)     |
| basename: removes directory      | names from pathnames             | basename(C)     |
| archive xdumpdir: prints the     | names of files on a backup       | xdumpdir(C)     |
| prints user and group IDs and    | names id:                        | id(C)           |
| user and group IDs and           | names id: print                  | id(ADM)         |
| isnanf: test for floating point  | NaN (Not-A-Number) /isnand,      | isnan(S)        |
| short interval                   | nap: Suspends execution for a    | nap(S)          |
| /nl_sscanf: converts formatted   | native language input            | nl_scanf(S)     |
| /nl_fprintf, nl_sprintf: formats | native language output           | nl_printf(S)    |
| nl_strcmp, nl_strncmp: compare   | native language strings          | nl_strcmp(S)    |
| operation nl_init: initializes   | native language support          | nl_init(S)      |
| access to a resource/ waitsem,   | nbwaitsem: awaits and checks     | waitsem(S)      |
| inode numbers                    | ncheck: generates names from     | ncheck(ADM)     |
| network                          | netutil: administers the UNIX    | netutil(ADM)    |
| administration nlsadmin:         | network listener service         | nlsadmin(ADM)   |
| netutil: administers the UNIX    | network                          | netutil(ADM)    |
| intro: introduction to the       | Network Services library         | intro(NSL)      |
| locally and over any supported   | network mmdf: routes mail        | mmdf(ADM)       |
| text file                        | newform: changes the format of a | newform(C)      |
| group                            | newgrp: logs user into a new     | newgrp(C)       |
| news: print                      | news items                       | news(C)         |
|                                  | news: print news items           | news(C)         |
| /fetch, store, delete, firstkey, | nextkey: performs database/      | dbm(S)          |
| free, _ffree,                    | _nfree: deallocate memory blocks | free(DOS)       |
| consistency check on/ _fheapchk, | _nheapchk: performs a minimal    | _fheapchk(DOS)  |
| of the next heap/ _fheapwalk,    | _nheapwalk: returns the address  | _fheapwalk(DOS) |
| tables nictable: process         | NIC database into channel/domain | nictable(ADM)   |
| process                          | nice: changes priority of a      | nice(S)         |
| different priority               | nice: runs a command at a        | nice(C)         |
| into channel/domain tables       | nictable: process NIC database   | nictable(ADM)   |
|                                  | nl: adds line numbers to a file  | nl(C)           |
| date and time                    | nl_asctime, nl_ctime: format     | nl_ctime(S)     |
| nl_asctime,                      | nl_ctime: format date and time   | nl_ctime(S)     |
| native language/ nl_printf,      | nl_fprintf, nl_sprintf: formats  | nl_printf(S)    |
| formatted native/ nl_scanf,      | nl_fscanf, nl_sscanf: converts   | nl_scanf(S)     |



|                                   |                                  |                  |
|-----------------------------------|----------------------------------|------------------|
| language support operation        | nl_init: initializes native      | nl_init(S)       |
| list                              | nlist: gets entries from name    | nlist(S)         |
| information                       | nl_langinfo: language            | nl_langinfo(S)   |
| nl_sprintf: formats native/       | nl_printf, nl_fprintf,           | nl_printf(S)     |
| service administration            | nlsadmin: network listener       | nlsadmin(ADM)    |
| converts formatted native/        | nl_scanf, nl_fscanf, nl_sscanf:  | nl_scanf(S)      |
| language/ nl_printf, nl_fprintf,  | nl_sprintf: formats native       | nl_printf(S)     |
| native/ nl_scanf, nl_fscanf,      | nl_sscanf: converts formatted    | nl_scanf(S)      |
| native language strings           | nl_strcmp, nl_strncmp: compare   | nl_strcmp(S)     |
| language strings nl_strcmp,       | nl_strncmp: compare native       | nl_strcmp(S)     |
|                                   | nm: prints name list             | nm(CP)           |
|                                   | nm: prints name list             | nm(XNX)          |
| malloc, _fmalloc,                 | _nmalloc: allocate main memory   | malloc(DOS)      |
| allocated/ _msize, _fmsize,       | _nmsize: return size of          | _msize(DOS)      |
| hangups and quits                 | nohup: runs a command immune to  | nohup(C)         |
| setjmp, longjmp: performs a       | nonlocal "goto"                  | setjmp(S)        |
| sigsetjmp, siglongjmp:            | non-local jumps                  | sigsetjmp(S)     |
| goodpw: check a password for      | non-obviousness                  | goodpw(ADM)      |
| false: returns with a             | nonzero exit value               | false(C)         |
| test for floating point NaN       | (Not-A-Number) /isnand, isnanf:  | isnan(S)         |
| terminal driving tables for       | nroff term:                      | term(F)          |
| null: the                         | null file                        | null(F)          |
|                                   | null: the null file              | null(F)          |
| /dodisk, lastlogin, monacct,      | nulladm, prctmp, prdaily/        | acctsh(ADM)      |
| factor: factor a                  | number                           | factor(C)        |
| /minor: returns major, new device | number, or minor device number   | major(K)         |
| random: generates a random        | number                           | random(C)        |
| rand, srand: generates a random   | number                           | rand(S)          |
| the/ /the list of major device    | numbers currently specified in   | majorsinuse(ADM) |
| nl: adds line                     | numbers to a file                | nl(C)            |
| ultoa: converts                   | numbers to characters            | ultoa(DOS)       |
| itoa: converts                    | numbers to integers              | itoa(DOS)        |
| atoi, atol: converts ASCII to     | numbers atof,                    | atof(S)          |
| generates names from inode        | numbers ncheck:                  | ncheck(ADM)      |
| library routines and error        | numbers /system services,        | Intro(S)         |
| a string to a double-precision    | number strtod, atof: converts    | strtod(S)        |
| numtbl: create a                  | numeric locale table             | numtbl(M)        |
| table                             | numtbl: create a numeric locale  | numtbl(M)        |
| dis:                              | object code disassembler         | dis(CP)          |
| DMD terminal wtnit:               | object downloader for the 5620   | wtnit(ADM)       |
| ldfcn: common                     | object file access routines      | ldfcn(F)         |
| mcs: manipulate the               | object file comment section      | mcs(CP)          |
| conv: common                      | object file converter            | conv(CP)         |
| cprs: compress a common           | object file                      | cprs(CP)         |
| dump: dump selected parts of an   | object file                      | dump(CP)         |
| ldopen, ldaopen: open a common    | object file for reading          | ldopen(S)        |
| line number entries of a common   | object file function /manipulate | ldread(S)        |
| size: prints the size of an       | object file                      | size(CP)         |
| size: prints the size of an       | object file                      | size(XNX)        |
| /retrieve symbol name for common  | object file symbol table entry   | ldgetname(S)     |
| syms: common                      | object file symbol table format  | syms(F)          |

|                                  |                                        |               |
|----------------------------------|----------------------------------------|---------------|
| section header of a common       | object file /an indexed/named . . .    | ldshread(S)   |
| section of a common              | object file /an indexed/named . . .    | ldsseek(S)    |
| ldaclose: close a common         | object file ldclose, . . . . .         | ldclose(S)    |
| read the file header of a common | object file ldhread: . . . . .         | ldhread(S)    |
| to the symbol table of a common  | object file ldtbseek: seek . . . .     | ldtbseek(S)   |
| line number entries in a common  | object file linenum: . . . . .         | linenum(F)    |
| C source listing from a common   | object file list: produce . . . . .    | list(CP)      |
| symbol table entry of a common   | object file /read an indexed . . . .   | ldtbread(S)   |
| information for a common         | object file reloc: relocation . . . .  | reloc(F)      |
| filehdr: file header for common  | object files . . . . .                 | filehdr(F)    |
| section header for a common      | object file scnhdr: . . . . .          | scnhdr(F)     |
| entries of a section of a common | object file /seek to line number . . . | ldlseek(S)    |
| entries of a section of a common | object file /seek to relocation . . .  | ldrseek(S)    |
| optional file header of a common | object file /seek to the . . . . .     | ldohseek(S)   |
| the printable strings in an      | object file strings: finds . . . . .   | strings(CP)   |
| a symbol table entry of a common | object file /the index of . . . . .    | ldtbindex(S)  |
| finds ordering relation for an   | object library loader: . . . . .       | loader(CP)    |
| driver entry points in a driver  | object module routines: finds . . . .  | routines(ADM) |
| 8086 Relocatable Format for      | object Modules 86rel: intel . . . .    | 86rel(F)      |
| copies characters between        | objects memmove: . . . . .             | memmove(DOS)  |
| copies characters between        | objects memmove: . . . . .             | memmove(S)    |
| number/ seed: getseed, setseed:  | obtain or set seed for random . . . .  | seed(S)       |
| a process until a signal         | occurs pause: Suspends . . . . .       | pause(S)      |
| od: displays files in            | octal format . . . . .                 | od(C)         |
| format                           | od: displays files in octal . . . . .  | od(C)         |
| fp_off, fp_seg: return           | offset and segment . . . . .           | fp_seg(DOS)   |
| FP_OFF, FP_SEG: get the          | offset and segment of an address . .   | fp_off(DOS)   |
| invokes a restricted version     | of red: . . . . .                      | ed(C)         |
| new file or rewrites an existing | one creat: creates a . . . . .         | creat(S)      |
| fopen:                           | opens a file . . . . .                 | fopen(DOS)    |
| writing open:                    | opens a file for reading or . . . . .  | open(DOS)     |
| and writing sopen:               | opens a file for shared reading . . .  | sopen(DOS)    |
| opensem:                         | opens a semaphore . . . . .            | opensem(S)    |
| fopen, freopen, fdopen:          | opens a stream . . . . .               | fopen(S)      |
| ev_open:                         | opens an event queue for input . . .   | ev_open(S)    |
| writing open:                    | opens file for reading or . . . . .    | open(S)       |
|                                  | opensem: opens a semaphore . . . . .   | opensem(S)    |
| system: executes an              | operating system command . . . . .     | system(DOS)   |
| prf:                             | operating system profiler . . . . .    | prf(HW)       |
| commands performed to stop the   | operating system rc0: run . . . . .    | rc0(ADM)      |
| native language support          | operation nl_init: initializes . . . . | nl_init(S)    |
| closedir: performs directory     | operations . . . . .                   | directory(S)  |
| msgctl: provides message control | operations . . . . .                   | msgctl(S)     |
| msgop: message                   | operations . . . . .                   | msgop(S)      |
| /strncpy, strnset: perform       | operations on characters in/ . . . . . | strncat(DOS)  |
| strdup, stricmp: perform         | operations on strings /strcsnp, . . .  | strcat(DOS)   |
| semctl: controls semaphore       | operations . . . . .                   | semctl(S)     |
| semop: performs semaphore        | operations . . . . .                   | semop(S)      |
| shmctl: controls shared memory   | operations . . . . .                   | shmctl(S)     |
| shmop: performs shared memory    | operations . . . . .                   | shmop(S)      |
| strdup: performs string          | operations . . . . .                   | string(S)     |



|                                  |                                  |                 |            |
|----------------------------------|----------------------------------|-----------------|------------|
| UNIX filesystems for             | optimal access time              | dcopy: copy     | dcopy(ADM) |
| vector getopt: gets              | option letter from argument      | getopt(S)       |            |
| object/ ldoheek: seek to the     | optional file header of a common | ldoheek(S)      |            |
| fcntl: file control              | options                          | fcntl(M)        |            |
| stty: Sets the                   | options for a terminal           | stty(C)         |            |
| t_optmgmt: manage                | options for a transport endpoint | t_optmgmt(NSL)  |            |
| getopt: parses command           | options                          | getopt(C)       |            |
| getoptcv - parse command         | options getopt: getopt,          | getopts(C)      |            |
| library lorder: finds            | ordering relation for an object  | lorder(CP)      |            |
| /acknowledge receipt of an       | orderly release indication       | t_rcvrel(NSL)   |            |
| t_sndrel: initiate an            | orderly release                  | t_sndrel(NSL)   |            |
| a directory, or a special or     | ordinary file mknod: makes       | mknod(S)        |            |
| os2ld:                           | OS/2 cross linker                | os2ld(CP)       |            |
|                                  | os2ld: OS/2 cross linker         | os2ld(CP)       |            |
| writes a byte to an I/O/ inb,    | outb: reads a byte from or       | inb(K)          |            |
| copies file archives in and      | out cpio:                        | cpio(C)         |            |
| word to a physical I/O/ ind,     | outd: reads, writes a 32-bit     | ind(K)          |            |
| dial: establishes an             | out-going terminal line/         | dial(S)         |            |
| port                             | outp: writes a byte to an output | outp(DOS)       |            |
| flushall: flushes all            | output buffers                   | flushall(DOS)   |            |
| ecvt, fcvt, gcvt: performs       | output conversions               | ecvt(S)         |            |
| cprintf: formats                 | output                           | cprintf(DOS)    |            |
| error: kernel error              | output device                    | error(M)        |            |
| tapedump: dumps magnetic tape to | output file                      | tapedump(C)     |            |
| /vsprintf: prints formatted      | output of a varargs/             | vsprintf(S)     |            |
| outp: writes a byte to an        | output port                      | outp(DOS)       |            |
| pr: prints files on the standard | output                           | pr(C)           |            |
| fwrite: writes to the            | output stream                    | fwrite(DOS)     |            |
| parameters sysdef:               | output values of tunable         | sysdef(ADM)     |            |
| of assembler and link editor     | output a.out: format             | a.out(F)        |            |
| buffered binary input and        | output fread, fwrite: performs   | fread(S)        |            |
| formats native language          | output /nl_fprintf, nl_sprintf:  | nl_printf(S)    |            |
| fprintf, sprintf: formats        | output printf,                   | printf(S)       |            |
| standard buffered input and      | output stdio: performs           | stdio(S)        |            |
| word from or to a physical/ inw, | outw: reads, writes a 16-bit     | inw(K)          |            |
| /acctdusg, accton, acctwtm -     | overview of accounting and/      | acct(ADM)       |            |
| /acctdusg, accton, acctwtm       | overview of accounting and/      | acct(ADM)       |            |
| creat: creates a new file or     | overwrites an existing one       | creat(DOS)      |            |
| purge:                           | overwrites specified files       | purge(C)        |            |
| chown: changes the               | owner and group of a file        | chown(S)        |            |
| chown: changes                   | owner ID                         | chown(C)        |            |
| quot: Summarizes filesystem      | ownership                        | quot(C)         |            |
| with ptrace for tracing a child/ | paccess: used in conjunction     | paccess(S)      |            |
| and expands files                | pack, pcatt, unpack: compresses  | pack(C)         |            |
| installpkg: install              | package                          | installpkg(ADM) |            |
| removepkg: remove installed      | package                          | removepkg(ADM)  |            |
| floating-point math              | package _fpreset: reinitializes  | _fpreset(DOS)   |            |
| interprocess communication       | package ftok: Standard           | stdipc(S)       |            |
| displaypkg: display installed    | packages                         | displaypkg(ADM) |            |
| sadc - system activity report    | package sar: sar, sa1, sa2,      | sar(ADM)        |            |
| xtt: extract and print xt driver | packet traces                    | xtt(ADM)        |            |



|                                   |                                      |               |
|-----------------------------------|--------------------------------------|---------------|
| pointer to block data             | paddr: returns virtual address . . . | paddr(K)      |
| Intro: lists manual               | page references . . . . .            | Intro(K)      |
| intro: lists manual               | page references . . . . .            | intro(K)      |
| between bytes and clicks (memory  | pages) /btoms, ctob: converts . .    | btoc(K)       |
| terminal 4014:                    | paginator for the TEKTRONIX 4014     | 4014(C)       |
| routines                          | panel: PANEL library . . . . .       | panel(S)      |
|                                   | panic: halts the system . . . . .    | panic(K)      |
| cmn_err: displays message or      | panics the system . . . . .          | cmn_err(K)    |
| mtune: tunable                    | parameter file . . . . .             | mtune(F)      |
| stune: local tunable              | parameter file . . . . .             | stune(F)      |
| to set value of a tunable         | parameter idtune: attempts . . .     | idtune(ADM)   |
| sysdef: output values of tunable  | parameters . . . . .                 | sysdef(ADM)   |
| gets process, process group, and  | parent process IDs /getppid: . .     | getpid(S)     |
| getopts: getopts, getoptcv -      | parse command options . . . . .      | getopts(C)    |
| getopts: getopts, getoptcv        | parse command options . . . . .      | getopts(C)    |
| getopt:                           | parses command options . . . . .     | getopt(C)     |
| fdisk: maintain disk              | partitions . . . . .                 | fdisk(ADM)    |
| dump: dump selected               | parts of an object file . . . . .    | dump(CP)      |
| files hdr: displays selected      | parts of executable binary . . . .   | hdr(CP)       |
| user space and the/ cpass,        | passc: passes character between .    | cpass(K)      |
| space and the/ cpass, passc:      | passes character between user . .    | cpass(K)      |
| password length                   | passlen: determine minimum . . .     | passlen(S)    |
| dialup shell password             | passwd: change login, group, or      | passwd(C)     |
|                                   | passwd: the password file . . . . .  | passwd(F)     |
| functions crypt:                  | password and file encryption . . .   | crypt(S)      |
| /putprpwnam: manipulate protected | password database entry . . . . .    | getprpwent(S) |
| putpwent: writes a                | password file entry . . . . .        | putpwent(S)   |
| setpwent, endpwent: gets          | password file entry /getpwnam, . .   | getpwent(S)   |
| passwd: the                       | password file . . . . .              | passwd(F)     |
| pwcheck: checks                   | password file . . . . .              | pwcheck(C)    |
| getpw: gets                       | password for a given user ID . . .   | getpw(S)      |
| goodpw: check a                   | password for non-obviousness . .     | goodpw(ADM)   |
| getpass: reads a                  | password . . . . .                   | getpass(S)    |
| getpasswd: read or clear a        | password . . . . .                   | getpasswd(S)  |
| /determine if                     | password is cryptic . . . . .        | accept_pw(S)  |
| passlen: determine minimum        | password length . . . . .            | passlen(S)    |
| login, group, or dialup shell     | password passwd: change . . . . .    | passwd(C)     |
| generate a pronounceable          | password randomword: . . . . .       | randomword(S) |
|                                   | paste: merges lines of files . . . . | paste(C)      |
| pathname variables                | pathconf: get configurable . . . .   | pathconf(S)   |
| directory getcwd: get the         | pathname of current working . . . .  | getcwd(S)     |
| pathconf: get configurable        | pathname variables . . . . .         | pathconf(S)   |
| delivers directory part of        | pathname dirname: . . . . .          | dirname(C)    |
| removes directory names from      | pathnames basename: . . . . .        | basename(C)   |
| searches for and processes a      | pattern in a file awk: . . . . .     | awk(C)        |
| fgrep: Searches a file for a      | pattern grep, egrep, . . . . .       | grep(C)       |
| a signal occurs                   | pause: Suspends a process until . .  | pause(S)      |
|                                   | pax: portable archive exchange . .   | pax(C)        |
| keyboard: the                     | pC keyboard . . . . .                | keyboard(HW)  |
| expands files pack,               | pcat, unpack: compresses and . .     | pack(C)       |
| a process popen,                  | pclose: initiates I/O to or from . . | popen(S)      |

|                                   |                                       |                   |
|-----------------------------------|---------------------------------------|-------------------|
| sigpending: examine               | pending signals . . . . .             | sigpending(S)     |
| reduction reduce:                 | perform audit data analysis and . .   | reduce(ADM)       |
| and/ fiecetomsbin, fmsbintoiee:   | perform conversions between IEEE      | fiecetomsbin(DOS) |
| dieeetomsbin, dmsbintoiee:        | perform conversions between MS/ .     | dieeetomsbin(DOS) |
| in/ /strnicmp, strncpy, strnset:  | perform operations on characters . .  | strncat(DOS)      |
| /strcpn, strdup, stricmp:         | perform operations on strings . . .   | strcat(DOS)       |
| environment rc2: run commands     | performed for multiuser . . . . .     | rc2(ADM)          |
| system rc0: run commands          | performed to stop the operating . .   | rc0(ADM)          |
| bsearch:                          | performs a binary search . . . . .    | bsearch(S)        |
| lfind, lsearch:                   | performs a linear array search . . .  | lfind(DOS)        |
| check on/ _fheapchk, _nheapchk:   | performs a minimal consistency . .    | _fheapchk(DOS)    |
| setjmp, longjmp:                  | performs a nonlocal "goto" . . . .    | setjmp(S)         |
| qsort:                            | performs a quicker sort . . . . .     | qsort(S)          |
| floor, fabs, ceil, fmod:          | performs absolute value, floor,/ . .  | floor(S)          |
| bessel, j0, j1, jn, y0, y1, yn:   | performs Bessel functions . . . . .   | bessel(S)         |
| and output fread, fwrite:         | performs buffered binary input . . .  | fread(S)          |
| /delete, firstkey, nextkey:       | performs database functions . . . .   | dbm(S)            |
| closedir:                         | performs directory operations . . .   | directory(S)      |
| exp, log, pow, sqrt, log10:       | performs exponential, logarithm,/ .   | exp(S)            |
| sinh, cosh, tanh:                 | performs hyperbolic functions . . .   | sinh(S)           |
| backup backup:                    | performs incremental filesystem . .   | backup(ADM)       |
| update lsearch, lfind:            | performs linear search and . . . . .  | lsearch(S)        |
| gamma:                            | performs log gamma function . . . .   | gamma(S)          |
| ecvt, fcvt, gcvt:                 | performs output conversions . . . .   | ecvt(S)           |
| system backups fsphoto:           | performs periodic semi-automated .    | fsphoto(ADM)      |
| functions curses:                 | performs screen and cursor . . . . .  | curses(S)         |
| semop:                            | performs semaphore operations . . .   | semop(S)          |
| operations shmop:                 | performs shared memory . . . . .      | shmop(S)          |
| and output stdio:                 | performs standard buffered input . .  | stdio(S)          |
| strdup:                           | performs string operations . . . . .  | string(S)         |
| /tgetflag, tgetstr, tgoto, tputs: | performs terminal functions . . . . . | termcap(S)        |
| tan, asin, acos, atan, atan2:     | performs trigonometric/ /cos, . . .   | trig(S)           |
| functions backup:                 | performs UNIX backup . . . . .        | backup(ADM)       |
| incremental filesystem/ xbackup:  | performs XENIX . . . . .              | xbackup(ADM)      |
| backups fsphoto: performs         | periodic semi-automated system . . .  | fsphoto(ADM)      |
| umask: sets the file              | permission mask . . . . .             | umask(DOS)        |
| chmod: changes file               | permissions . . . . .                 | chmod(DOS)        |
| permissions: format of UUCP       | permissions file . . . . .            | permissions(F)    |
| check the uucp directories and    | permissions file uucheck: . . . . .   | uucheck(ADM)      |
| locking: sets read and write      | permissions for a portion of a/ . . . | locking(DOS)      |
| permissions file                  | permissions: format of UUCP . . . .   | permissions(F)    |
| chmod: changes the access         | permissions of a file or/ . . . . .   | chmod(C)          |
| to a terminal msg:                | permits or denies messages sent . .   | msg(C)            |
| acct: format of                   | per-process accounting file . . . . . | acct(F)           |
| acctcms: command summary from     | per-process accounting records . . .  | acctcms(ADM)      |
| errno: Sends system error/        | perror, sys_errlist, sys_nerr, . . .  | perror(S)         |
| drivers physio,                   | physck: raw I/O for block . . . . .   | physio(K)         |
| copies bytes to and from a        | physical address copyio: . . . . .    | copyio(K)         |
| ptok, ktop: converts virtual and  | physical addresses . . . . .          | ptok(K)           |
| convert a virtual address to a    | physical address vtop: . . . . .      | vtop(K)           |
| reads, writes a 32-bit word to a  | physical I/O address ind, outd: . . . | ind(K)            |



|                                   |                                        |                |
|-----------------------------------|----------------------------------------|----------------|
| a 16-bit word from or to a        | physical I/O address /writes . . .     | inw(K)         |
| db_read: transfers data from      | physical memory to a user/ . . .       | db_read(K)     |
| /db_free: allocates and frees     | physically contiguous memory . . .     | db_alloc(K)    |
| block drivers                     | physio, physck: raw I/O for . . .      | physio(K)      |
| split: Splits a file into         | pieces . . . . .                       | split(C)       |
| programmed I/O requests           | pio_breakup: breaks up . . . . .       | pio_breakup(K) |
| pipe                              | pipe: creates an interprocess . . .    | pipe(S)        |
| pipe: creates an interprocess     | pipe . . . . .                         | pipe(S)        |
| tee: creates a tee in a           | pipe . . . . .                         | tee(C)         |
| data in memory                    | plock: lock process, text, or . . .    | plock(S)       |
| subroutines                       | plot: graphics interface . . . . .     | plot(F)        |
| images                            | plot: graphics interface . . . . .     | plot(S)        |
| sterror: gets error message       | pnch: file format for card . . . . .   | pnch(F)        |
| sterror: gets error message       | pointer from last routine call/ . . .  | sterror(DOS)   |
| fseek: moves a file               | pointer from last routine call/ . . .  | sterror(S)     |
| rewind: repositions a file        | pointer . . . . .                      | fseek(DOS)     |
| lseek: moves read/write file      | pointer in a stream /ftell, . . . . .  | fseek(S)       |
| paddr: returns virtual address    | pointer . . . . .                      | lseek(S)       |
| the current position of a file    | pointer to block data . . . . .        | paddr(K)       |
| changes the position of a file    | pointer ftell: finds . . . . .         | ftell(DOS)     |
| the current position of the file  | pointer lseek: . . . . .               | lseek(DOS)     |
| routines: finds driver entry      | pointer tell: gets . . . . .           | tell(DOS)      |
| utility purge(C) purge: the       | points in a driver object/ . . . . .   | routines(ADM)  |
| poll: format of UUCP              | policy file of the sanitization . . .  | purge(F)       |
| input/output multiplexing         | poll file . . . . .                    | poll(F)        |
| a buffer from the block buffer    | poll: format of UUCP Poll file . . .   | poll(F)        |
| queue ev_pop:                     | poll: STREAMS . . . . .                | poll(S)        |
| or from a process                 | pool geteblk, getblk: gets . . . . .   | geteblk(K)     |
| outp: writes a byte to an output  | pop the next event off the . . . . .   | ev_pop(S)      |
| pax:                              | popen, pclose: initiates I/O to . . .  | popen(S)       |
| read and write permissions for a  | port . . . . .                         | outp(DOS)      |
| , tty2[A-H]: interface to serial  | portable archive exchange . . . . .    | pax(C)         |
| machine is suitable for security  | portion of a file locking: sets . . .  | locking(DOS)   |
| exponential,/ exp, log,           | ports /, tty1[A-H], tty2[a-h] . . .    | serial(HW)     |
| /Performs exponential, logarithm, | port /verify . . . . .                 | check_data(S)  |
| output                            | pow, sqrt, log10: performs . . . . .   | exp(S)         |
| /lastlogin, monacct, nulladm,     | power, square root functions . . . .   | exp(S)         |
| /monacct, nulladm, prtmp,         | pr: prints files on the standard . . . | pr(C)          |
| dc: invokes an arbitrary          | prctmp, prdaily, prtacct,/ . . . . .   | acctsh(ADM)    |
| monitor:                          | prdaily, prtacct, runacct,/ . . . . .  | acctsh(ADM)    |
| cpp: the C language               | precision calculator . . . . .         | dc(C)          |
| unget: undoes a                   | prepares execution profile . . . . .   | monitor(S)     |
| dma_rlse: releases                | preprocessor . . . . .                 | cpp(CP)        |
| _expand: changes the size of a    | previous get of an SCCS file . . . . . | unget(CP)      |
| sptfree: releases memory          | previously allocated DMA channel . .   | dma_rlse(K)    |
| profiler: prfld, prfstat,         | previously allocated memory/ . . . .   | _expand(DOS)   |
| prfpr - UNIX/ profiler:           | previously allocated with/ . . . . .   | sptfree(K)     |
| /prfld, prfstat, prfdc, prfsnap,  | prf: operating system profiler . . . . | prf(HW)        |
| prfpr - UNIX system/ . . . . .    | prfdc, prfsnap, prfpr -/ . . . . .     | profiler(ADM)  |
|                                   | prfld, prfstat, prfdc, prfsnap, . . .  | profiler(ADM)  |
|                                   | prfpr - UNIX system/ . . . . .         | profiler(ADM)  |



|                                  |                                                |                   |
|----------------------------------|------------------------------------------------|-------------------|
| profiler: prfld, prfstat, prfdc, | prfsnap, prfpr - UNIX/ . . . . .               | profiler(ADM)     |
| -/ profiler: prfld,              | prfstat, prfdc, prfsnap, prfpr . . . . .       | profiler(ADM)     |
| lock: locks a process in         | primary memory . . . . .                       | lock(S)           |
| graphical files gps: graphical   | primitive string, format of . . . . .          | gps(F)            |
| types:                           | primitive system data types . . . . .          | types(F)          |
| printf:                          | print a message on the console . . . . .       | printf(K)         |
| temporarily auths:               | print and/or restrict authorizations . . . . . | auths(C)          |
| to a serial/ consoleprint:       | print file to printer attached . . . . .       | consoleprint(ADM) |
| news:                            | print news items . . . . .                     | news(C)           |
| infocmp: compare or              | print out terminfo descriptions . . . . .      | infocmp(ADM)      |
| utility lpsh: menu driven lp     | print service administration . . . . .         | lpsh(ADM)         |
| filters used with the LP         | print service /administer . . . . .            | lpfilter(ADM)     |
| forms used with the LP           | print service /administer . . . . .            | lpforms(ADM)      |
| jwin:                            | print size of layer . . . . .                  | jwin(C)           |
| the user's terminal lprint:      | print to a printer attached to . . . . .       | lprint(C)         |
| and names id:                    | print user and group IDs . . . . .             | id(ADM)           |
| xtt: extract and                 | print xt driver packet traces . . . . .        | xtt(ADM)          |
| xts: extract and                 | print xt driver statistics . . . . .           | xts(ADM)          |
| file strings: finds the          | printable strings in an object . . . . .       | strings(CP)       |
| initialization message           | printcfg: displays driver . . . . .            | printcfg(K)       |
| consoleprint: print file to      | printer attached to a serial/ . . . . .        | consoleprint(ADM) |
| terminal lprint: print to a      | printer attached to the user's . . . . .       | lprint(C)         |
| lp, lp0, lp1, lp2: line          | printer device interfaces . . . . .            | lp(HW)            |
| disable: turns off terminals and | printers . . . . .                             | disable(C)        |
| turns on terminals and line      | printers enable: . . . . .                     | enable(C)         |
| formats output                   | printf, fprintf, sprintf: . . . . .            | printf(S)         |
| console                          | printf: print a message on the . . . . .       | printf(K)         |
| lpusers: set                     | printing queue priorities . . . . .            | lpusers(ADM)      |
| cal:                             | prints a calendar . . . . .                    | cal(C)            |
| console putchar:                 | prints a character on the . . . . .            | putchar(K)        |
| the console devern:              | prints a device error message on . . . . .     | devern(K)         |
| prs:                             | prints an SCCS file . . . . .                  | prs(CP)           |
| sddate:                          | prints and sets backup dates . . . . .         | sddate(C)         |
| date:                            | prints and sets the date . . . . .             | date(C)           |
| activity sact:                   | prints current SCCS file editing . . . . .     | sact(CP)          |
| output pr:                       | prints files on the standard . . . . .         | pr(C)             |
| vprintf, fprintf, vsprintf:      | prints formatted output of a/ . . . . .        | vprintf(S)        |
| banner:                          | prints large letters . . . . .                 | banner(C)         |
| information lpstat:              | prints lineprinter status . . . . .            | lpstat(C)         |
| nm:                              | prints name list . . . . .                     | nm(CP)            |
| nm:                              | prints name list . . . . .                     | nm(XNX)           |
| filesystem fsname:               | prints or changes the name of a . . . . .      | fsname(ADM)       |
| acctcom: Searches for and        | prints process accounting files . . . . .      | acctcom(ADM)      |
| messages strace:                 | prints STREAMS trace . . . . .                 | strace(ADM)       |
| yes:                             | prints string repeatedly . . . . .             | yes(C)            |
| stream head:                     | prints the first few lines of a . . . . .      | head(C)           |
| UNIX system uname:               | prints the name of the current . . . . .       | uname(C)          |
| backup archive xdumpdir:         | prints the names of files on a . . . . .       | xdumpdir(C)       |
| file size:                       | prints the size of an object . . . . .         | size(CP)          |
| file size:                       | prints the size of an object . . . . .         | size(XNX)         |
| names id:                        | prints user and group IDs and . . . . .        | id(C)             |

|                                  |                                       |               |
|----------------------------------|---------------------------------------|---------------|
| pwd:                             | prints working directory name . . .   | pwd(C)        |
| lpusers: set printing queue      | priorities . . . . .                  | lpusers(ADM)  |
| nice: changes                    | priority of a process . . . . .       | nice(S)       |
| runs a command at a different    | priority nice: . . . . .              | nice(C)       |
| process getpriv: get system      | privileges associated with this . . . | getpriv(S)    |
| setpriv: set system              | privileges for this process . . . .   | setpriv(S)    |
| /startup, turnacct - shell       | procedures for accounting . . . .     | acctsh(ADM)   |
| - system initialization          | procedures brc: brc, bcheckrc . . .   | brc(ADM)      |
| abort: terminates a              | process . . . . .                     | abort(DOS)    |
| acctprc: acctprc1, acctprc2 -    | process accounting . . . . .          | acctprc(ADM)  |
| acctprc: acctprc1, acctprc2      | process accounting . . . . .          | acctprc(ADM)  |
| acct: enables or disables        | process accounting . . . . .          | acct(S)       |
| acctcom: Searches for and prints | process accounting files . . . . .    | acctcom(ADM)  |
| alarm: Sets a                    | process' alarm clock . . . . .        | alarm(S)      |
| times: gets                      | process and child process times . .   | times(S)      |
| atexit: calls a                  | process at termination . . . . .      | atexit(DOS)   |
| atexit: calls a                  | process at termination . . . . .      | atexit(S)     |
| init, inir:                      | process control initialization . . .  | init(M)       |
| timex: time a command; report    | process data and system activity . .  | timex(ADM)    |
| deliver: MMDF mail delivery      | process . . . . .                     | deliver(ADM)  |
| time delay: delays               | process execution for specified . .   | delay(K)      |
| exit: terminates the calling     | process . . . . .                     | exit(DOS)     |
| exit, _exit: terminates a        | process . . . . .                     | exit(S)       |
| fork: creates a new              | process . . . . .                     | fork(S)       |
| /getpgpr, getppid: gets process, | process group, and parent/ . . . .    | getpid(S)     |
| setpgid: set                     | process group ID for job control . .  | setpgid(S)    |
| tcgetpgpr, tcsetpgpr:            | process group id functions . . . .    | tcpgrp(S)     |
| setpgpr: Sets                    | process group ID . . . . .            | setpgpr(S)    |
| setsid: create session and set   | process ID . . . . .                  | setsid(S)     |
| getpid: returns a                | process identification number . . .   | getpid(DOS)   |
| process group, and parent        | process IDs /Gets process, . . . .    | getpid(S)     |
| lock: locks a                    | process in primary memory . . . .     | lock(S)       |
| kill: terminates a               | process . . . . .                     | kill(C)       |
| channel/domain tables nictable:  | process NIC database into . . . . .   | nictable(ADM) |
| nice: changes priority of a      | process . . . . .                     | nice(S)       |
| kill: Sends a signal to a        | process or a group of processes . .   | kill(S)       |
| getpid, getpgpr, getppid: gets   | process, process group, and/ . . . .  | getpid(S)     |
| psignal: sends signal to a       | process . . . . .                     | psignal(K)    |
| ptrace: traces a                 | process . . . . .                     | ptrace(S)     |
| signal: sends a signal to a      | process . . . . .                     | signal(K)     |
| spawnl, spawnvp: creates a new   | process . . . . .                     | spawn(DOS)    |
| ps: reports                      | process status . . . . .              | ps(C)         |
| memory plock: lock               | process, text, or data in . . . . .   | plock(S)      |
| times: gets process and child    | process times . . . . .               | times(S)      |
| wait: waits for a child          | process to stop or terminate . . . .  | wait(S)       |
| pause: Suspends a                | process until a signal occurs . . . . | pause(S)      |
| sigsem: Signals a                | process waiting on a semaphore . .    | sigsem(S)     |
| wakeup: wakes up a sleeping      | process . . . . .                     | wakeup(K)     |
| checklist: list of filesystems   | processed by fsck . . . . .           | checklist(F)  |
| awk: Searches for and            | processes a pattern in a file . . . . | awk(C)        |
| killall: kill all active         | processes . . . . .                   | killall(ADM)  |



|                                  |                                        |                 |
|----------------------------------|----------------------------------------|-----------------|
| tty device canon:                | processes raw input data from . . .    | canon(K)        |
| to a process or a group of       | processes kill: Sends a signal . . .   | kill(S)         |
| awaits completion of background  | processes wait: . . . . .              | wait(C)         |
| execvpe: load and execute a      | process /execv, execve, execvp, . . .  | exec(DOS)       |
| privileges associated with this  | process getpriv: get system . . .      | getpriv(S)      |
| shutdown: terminates all         | processing . . . . .                   | shutdown(ADM)   |
| sleep: suspends                  | processing temporarily . . . . .       | sleep(K)        |
| list: list                       | processor channel for MMDF . . .       | list(ADM)       |
| m4: invokes a macro              | processor . . . . .                    | m4(CP)          |
| machid: machid, i386 - get       | processor type truth value . . . .     | machid(C)       |
| initiates I/O to or from a       | process popen, pclose: . . . . .       | popen(S)        |
| set system privileges for this   | process setpriv: . . . . .             | setpriv(S)      |
| with ptrace for tracing a child  | process /used in conjunction . . .     | paccess(S)      |
| authentication/ authaudit:       | produce audit records due to . . .     | authaudit(S)    |
| subsystem events dlvr_audit:     | produce audit records for . . . .      | dlvr_audit(ADM) |
| common object file list:         | produce C source listing from a . .    | list(CP)        |
| t_error:                         | produce error message . . . . .        | t_error(NSL)    |
| modifications to the/ swconfig:  | produces a list of the software . .    | swconfig(C)     |
|                                  | prof: displays profile data . . . .    | prof(CP)        |
|                                  | prof: displays profile data . . . .    | prof(XNX)       |
|                                  | prof: profile within a function . .    | prof(M)         |
| time profile                     | profil: creates an execution . . .     | profil(S)       |
| prof: displays                   | profile data . . . . .                 | prof(CP)        |
| prof: displays                   | profile data . . . . .                 | prof(XNX)       |
| line-by-line execution count     | profile data lprof: display . . . .    | lprof(CP)       |
| monitor: prepares execution      | profile . . . . .                      | monitor(S)      |
| at login time                    | profile: Sets up an environment . .    | profile(M)      |
| prof:                            | profile within a function . . . . .    | prof(M)         |
| creates an execution time        | profile profil: . . . . .              | profil(S)       |
| prf: operating system            | profiler . . . . .                     | prf(HW)         |
| prfsnap, prfpr - UNIX/           | profiler: prfld, prfstat, prfdc, . . . | profiler(ADM)   |
| prfpr - UNIX system              | profiler /prfdc, prfsnap, . . . . .    | profiler(ADM)   |
| assert: helps verify validity of | program . . . . .                      | assert(S)       |
| boot: uNIX boot                  | program . . . . .                      | boot(HW)        |
| tape: magnetic tape maintenance  | program . . . . .                      | tape(C)         |
| etext, edata: last locations in  | program end, . . . . .                 | end(S)          |
| pio_breakup: breaks up           | programmed I/O requests . . . . .      | pio_breakup(K)  |
| cb: beautifies C                 | programs . . . . .                     | cb(CP)          |
| lex: generates                   | programs for lexical analysis . . .    | lex(CP)         |
| alloca: allocates bytes from the | program's stack . . . . .              | alloca(DOS)     |
| xref: cross-references C         | programs . . . . .                     | xref(CP)        |
| xstr: extracts strings from C    | programs . . . . .                     | xstr(CP)        |
| attributes of files and          | programs /check discretionary . .      | discr(S)        |
| and regenerates groups of        | programs /Maintains, updates, . .      | make(CP)        |
| domain of a program              | promain: restrict the execution . .    | promain(M)      |
| day asktime:                     | prompts for the correct time of . .    | asktime(ADM)    |
| randomword: generate a           | pronounceable password . . . . .       | randomword(S)   |
| entry /putprpwnam: manipulate    | protected password database . . .      | getprpwent(S)   |
|                                  | proto: prototype job file for at . .   | proto(ADM)      |
| windowing terminal/ layers:      | protocol used between host and . .     | layers(M)       |
| xtproto: multiplexed channels    | protocol used by xt(HW)/ . . . .       | xtproto(M)      |



|                                   |                                                                                      |                                                                                                                 |                                                 |
|-----------------------------------|--------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|-------------------------------------------------|
| information                       | t_getinfo: get                                                                       | protocol-specific service . . . . .                                                                             | t_getinfo(NSL)                                  |
|                                   | proto:                                                                               | prototype job file for at . . . . .                                                                             | proto(ADM)                                      |
|                                   | labelit:                                                                             | provide labels for filesystems . . . . .                                                                        | labelit(ADM)                                    |
| locking on files                  | lockf:                                                                               | provide semaphores and record . . . . .                                                                         | lockf(S)                                        |
| operations                        | msgctl:                                                                              | provides message control . . . . .                                                                              | msgctl(S)                                       |
|                                   | prs:                                                                                 | prints an SCCS file . . . . .                                                                                   | prs(CP)                                         |
| /nulladm, prctmp, prdaily,        | prtacct, runacct, shutacct,/<br>ps:                                                  | reports process status . . . . .                                                                                | acctsh(ADM)<br>ps(C)                            |
|                                   | sxt:                                                                                 | pseudo-device driver . . . . .                                                                                  | sxt(M)                                          |
| process                           | psignal:                                                                             | sends signal to a . . . . .                                                                                     | psignal(K)                                      |
| information                       | pstat:                                                                               | reports system . . . . .                                                                                        | pstat(C)                                        |
| physical addresses                | ptok, ktok:                                                                          | converts virtual and . . . . .                                                                                  | ptok(K)                                         |
| /used in conjunction with         | ptrace:                                                                              | traces a process . . . . .                                                                                      | ptrace(S)                                       |
|                                   | purge:                                                                               | overwrites specified . . . . .                                                                                  | purge(C)                                        |
| sanitization utility              | purge(C)                                                                             | purge: the policy file of the . . . . .                                                                         | purge(F)                                        |
| file of the sanitization utility  | purge(C)                                                                             | purge: the policy . . . . .                                                                                     | purge(F)                                        |
| stream                            | ungetc:                                                                              | pushes character back into input . . . . .                                                                      | ungetc(S)                                       |
|                                   | write to clists                                                                      | putc, putcb, putcbp, putcf: . . . . .                                                                           | putc(K)                                         |
| a character or word on a/         | putc, putcb, putcbp, putcf:                                                          | puts . . . . .                                                                                                  | putc(S)                                         |
| clists                            | putc, putcb, putcbp, putcf:                                                          | write to . . . . .                                                                                              | putc(K)                                         |
| putc, putcb,                      | putcbp, putcf:                                                                       | write to clists . . . . .                                                                                       | putc(K)                                         |
| putc, putcb, putcbp,              | putcf:                                                                               | write to clists . . . . .                                                                                       | putc(K)                                         |
| console                           | putch:                                                                               | writes a character to the . . . . .                                                                             | putch(DOS)                                      |
| character or word on a/           | putc, putchar, fputc, putw:                                                          | puts a . . . . .                                                                                                | putc(S)                                         |
| the console                       | putchar:                                                                             | prints a character on . . . . .                                                                                 | putchar(K)                                      |
| /setdvagent, enddvagent,          | putdvagent, copydvagent:/                                                            | . . . . .                                                                                                       | getdvagent(S)                                   |
| environment                       | putenv:                                                                              | changes or adds value to . . . . .                                                                              | putenv(S)                                       |
| stream                            | putmsg:                                                                              | send a message on a . . . . .                                                                                   | putmsg(S)                                       |
| control/ /setprcment, endprcment, | putprcment:                                                                          | manipulate command . . . . .                                                                                    | getprcment(S)                                   |
| control/ /setprdfent, endprdfent, | putprdfent:                                                                          | manipulate default . . . . .                                                                                    | getprdfent(S)                                   |
| control/ /setprfient, endprfient, | putprfient:                                                                          | manipulate file . . . . .                                                                                       | getprfient(S)                                   |
| /setprpwent, endprpwent,          | putprpwent:                                                                          | manipulate protected/ . . . . .                                                                                 | getprpwent(S)                                   |
| control/ /setprtcent, endprtcent, | putprtcent:                                                                          | manipulate terminal . . . . .                                                                                   | getprtcent(S)                                   |
| entry                             | putpwent:                                                                            | writes a password file . . . . .                                                                                | putpwent(S)                                     |
| putc, putchar, fputc, putw:       | puts:                                                                                | a character or word on a/ . . . . .                                                                             | putc(S)                                         |
| puts, fputs:                      | puts:                                                                                | a string on a stream . . . . .                                                                                  | puts(S)                                         |
| cputs:                            | puts:                                                                                | a string to the console . . . . .                                                                               | cputs(DOS)                                      |
| stream                            | puts, fputs:                                                                         | puts a string on a . . . . .                                                                                    | puts(S)                                         |
| on a/                             | putc, putchar, fputc,                                                                | putw:                                                                                                           | puts a character or word . . . . .              |
|                                   | pwcheck:                                                                             | checks password file . . . . .                                                                                  | pwcheck(C)                                      |
| name                              | pwd:                                                                                 | prints working directory . . . . .                                                                              | pwd(C)                                          |
| pw_mapping: pw_nametoid,          | pw_idtoname, gr_nametoid,/<br>pw_idtoname, gr_nametoid,/<br>gr_nametoid, pw_mapping: | pw_mapping: pw_nametoid, . . . . .<br>pw_mapping: pw_nametoid, . . . . .<br>pw_nametoid, pw_idtoname, . . . . . | pw_mapping(S)<br>pw_mapping(S)<br>pw_mapping(S) |
| tapecntl: AT&T tape control for   | QIC-24/QIC-02 tape device                                                            | . . . . .                                                                                                       | tapecntl(C)                                     |
|                                   | qsort:                                                                               | performs a quicker sort . . . . .                                                                               | qsort(S)                                        |
| tput:                             | queries the terminfo database                                                        | . . . . .                                                                                                       | tput(C)                                         |
| ev_close: close the event         | queue and all associated/                                                            | . . . . .                                                                                                       | ev_close(S)                                     |
| ev_block: wait until the          | queue contains an event                                                              | . . . . .                                                                                                       | ev_block(S)                                     |
| ev_resume: restart a suspended    | queue                                                                                | . . . . .                                                                                                       | ev_resume(S)                                    |

|                                  |                                            |                                         |
|----------------------------------|--------------------------------------------|-----------------------------------------|
| ev_suspend: suspends an event    | queue . . . . .                            | ev_suspend(S)                           |
| ev_suspend: Suspends an event    | queue . . . . .                            | ev_susp(S)                              |
| transit queue: MMDF              | queue files for storing mail in . . . . .  | queue(ADM)                              |
| ev_open: opens an event          | queue for input . . . . .                  | ev_open(S)                              |
| storing mail in transit          | queue: MMDF queue files for . . . . .      | queue(ADM)                              |
| msgget: gets message             | queue . . . . .                            | msgget(S)                               |
| lpusers: set printing            | queue priorities . . . . .                 | lpusers(ADM)                            |
| ipcrm: removes a message         | queue, semaphore set or shared/            | ipcrm(ADM)                              |
| checkqueue: MMDF                 | queue status report generator . . . . .    | checkqueue(ADM)                         |
| block I/O request to a device's  | queue disksort: adds a . . . . .           | disksort(K)                             |
| all events currently in the      | queue ev_flush: discard . . . . .          | ev_flush(S)                             |
| list of devices feeding an event | queue ev_getdev: gets a . . . . .          | ev_getdev(S)                            |
| pop the next event off the       | queue ev_pop: . . . . .                    | ev_pop(S)                               |
| read the next event in the       | queue ev_read: . . . . .                   | ev_read(S)                              |
| of events currently in the       | queue /Returns the number . . . . .        | ev_count(S)                             |
| dma_start:                       | queues DMA request . . . . .               | dma_start(K)                            |
| qsort: performs a                | quicker sort . . . . .                     | qsort(S)                                |
| a command immune to hangups and  | quits nohup: runs . . . . .                | nohup(C)                                |
| ownership                        | quot: Summarizes filesystem . . . . .      | quot(C)                                 |
| execution program                | raise: send signal <i>sig</i> to . . . . . | raise(S)                                |
| executing program                | raise: sends a signal to the . . . . .     | raise(DOS)                              |
| number                           | rand, srand: generates a random . . . . .  | rand(S)                                 |
| number                           | random: generates a random . . . . .       | random(C)                               |
| ranlib: converts archives to     | random libraries . . . . .                 | ranlib(CP)                              |
| /setseed: obtain or set seed for | random number generator . . . . .          | seed(S)                                 |
| random: generates a              | random number . . . . .                    | random(C)                               |
| rand, srand: generates a         | random number . . . . .                    | rand(S)                                 |
| pronounceable password           | randomword: generate a . . . . .           | randomword(S)                           |
| random libraries                 | ranlib: converts archives to . . . . .     | ranlib(CP)                              |
| cfsetispeed, cfsetospeed: baud   | rate functions /cfgetospeed, . . . . .     | cfspeed(S)                              |
| FORTRAN into standard FORTRAN    | ratfor: converts Rational . . . . .        | ratfor(CP)                              |
| FORTRAN ratfor: converts         | rational FORTRAN into standard . . . . .   | ratfor(CP)                              |
| canon: processes                 | raw input data from tty device . . . . .   | canon(K)                                |
| physio, physck:                  | raw I/O for block drivers . . . . .        | physio(K)                               |
| stop the operating system        | rc0: run commands performed to . . . . .   | rc0(ADM)                                |
| multiuser environment            | rc2: run commands performed for . . . . .  | rc2(ADM)                                |
|                                  | rcc: AT&T C compiler . . . . .             | rcc(CP)                                 |
|                                  | systems                                    | rcp: copies files across UNIX . . . . . |
| data to be read                  | rdchk: checks to see if there is . . . . . | rdchk(S)                                |
| entry of a common/ ldtbread:     | read an indexed symbol table . . . . .     | ldtbread(S)                             |
| header of/ ldshread, ldnsbread:  | read an indexed/named section . . . . .    | ldshread(S)                             |
| portion of a file locking: sets  | read and write permissions for a . . . . . | locking(DOS)                            |
| generated by the audit/ auditd:  | read audit collection files . . . . .      | auditd(ADM)                             |
| getc, getcb, getcbp, getcf:      | read clist buffers . . . . .               | getc(K)                                 |
| in a file getdents:              | read directory entries and put . . . . .   | getdents(S)                             |
| specifications idmkinit:         | read files containing . . . . .            | idmkinit(ADM)                           |
| setlocale: Set or                | read international environment . . . . .   | setlocale(S)                            |
| getpasswd:                       | read or clear a password . . . . .         | getpasswd(S)                            |
|                                  | read: reads from a file . . . . .          | read(DOS)                               |
|                                  | read: reads from a file . . . . .          | read(S)                                 |
| member of an archive/ ldahread:  | read the archive header of a . . . . .     | ldahread(S)                             |



|                                 |                         |                                                       |                   |
|---------------------------------|-------------------------|-------------------------------------------------------|-------------------|
| information                     | hwconfig:               | read the configuration . . . . .                      | hwconfig(ADM)     |
| object file                     | ldfhead:                | read the file header of a common . . . . .            | ldfhead(S)        |
| queue                           | ev_read:                | read the next event in the . . . . .                  | ev_read(S)        |
| sopen:                          | opens a file for shared | reading and writing . . . . .                         | sopen(DOS)        |
| open:                           | opens a file for        | reading or writing . . . . .                          | open(DOS)         |
| open:                           | opens file for          | reading or writing . . . . .                          | open(S)           |
| or unlocks a file region for    |                         | reading or writing /Locks . . . . .                   | locking(S)        |
| open a common object file for   |                         | reading ldopen, ldaopen: . . . . .                    | ldopen(S)         |
| to see if there is data to be   |                         | read rdchk: checks . . . . .                          | rdchk(S)          |
| byte to an I/O/                 | inb, outb:              | reads a byte from or writes a . . . . .               | inb(K)            |
|                                 | getpass:                | reads a password . . . . .                            | getpass(S)        |
| /repoutsb, repoutsw, repoutsd:  |                         | reads and writes streams of/ . . . . .                | repins(K)         |
|                                 | defopen, defread:       | reads default entries . . . . .                       | defopen(S)        |
|                                 | read:                   | reads from a file . . . . .                           | read(DOS)         |
|                                 | read:                   | reads from a file . . . . .                           | read(S)           |
|                                 | fread:                  | reads from the input stream . . . . .                 | fread(DOS)        |
|                                 | line:                   | reads one line . . . . .                              | line(C)           |
|                                 | mail:                   | Sends, reads or disposes of mail . . . . .            | mail(C)           |
| or to a physical I/O/           | inw, outw:              | reads, writes a 16-bit word from . . . . .            | inw(K)            |
| physical I/O address            | ind, outd:              | reads, writes a 32-bit word to a . . . . .            | ind(K)            |
|                                 | lseek:                  | moves read/write file pointer . . . . .               | lseek(S)          |
| memory                          | malloc, free,           | realloc, calloc: allocates main . . . . .             | malloc(S)         |
| getclk:                         | gets string from        | real-time clock . . . . .                             | getclk(M)         |
|                                 | clock:                  | the system real-time (time of day) clock . . . . .    | clock(F)          |
| setclock:                       | Sets the system         | real-time (time of day) clock . . . . .               | setclock(ADM)     |
| systems and shuts down/         | haltsys,                | reboot: closes out the file . . . . .                 | haltsys(ADM)      |
| the action to be taken upon     |                         | receipt of a signal /defines . . . . .                | signal(DOS)       |
| specifies what to do upon       |                         | receipt of a signal signal: . . . . .                 | signal(S)         |
|                                 | t_rcvrel:               | acknowledge receipt of an orderly release/ . . . . .  | t_rcvrel(NSL)     |
|                                 | t_rcvudata:             | receive a data unit . . . . .                         | t_rcvudata(NSL)   |
|                                 | indication t_rcvuderr:  | receive a unit data error . . . . .                   | t_rcvuderr(NSL)   |
| sent over a connection          | t_rcv:                  | receive data or expedited data . . . . .              | t_rcv(NSL)        |
| connect request                 | t_rcvconnect:           | receive the confirmation from a . . . . .             | t_rcvconnect(NSL) |
|                                 | rmail:                  | submit remote mail received via UUCP . . . . .        | rmail(ADM)        |
| lineprinters                    | lpinit:                 | adds, reconfigures and maintains . . . . .            | lpinit(ADM)       |
| access audit session data on a  |                         | record basis /open and . . . . .                      | audit(S)          |
| lockf:                          | provide semaphores and  | record locking on files . . . . .                     | lockf(S)          |
| events                          | authaudit:              | produce audit records due to authentication . . . . . | authaudit(S)      |
|                                 | dlvr_audit:             | produce audit records for subsystem events . . . . .  | dlvr_audit(ADM)   |
| from per-process accounting     |                         | records /command summary . . . . .                    | acctcms(ADM)      |
| manipulate connect accounting   |                         | records fwtmpt: fwtmpt, wtmptfix: . . . . .           | fwtmpt(ADM)       |
|                                 | version of              | red: invokes a restricted . . . . .                   | ed(C)             |
| analysis and reduction          |                         | reduce: perform audit data . . . . .                  | reduce(ADM)       |
| perform audit data analysis and |                         | reduction reduce: . . . . .                           | reduce(ADM)       |
|                                 | Intro:                  | lists manual page references . . . . .                | Intro(K)          |
|                                 | intro:                  | lists manual page references . . . . .                | intro(K)          |
| regular expressions             | regex,                  | regcmp: compiles and executes . . . . .               | regex(S)          |
|                                 | expressions             | regcmp: compiles regular . . . . .                    | regcmp(CP)        |
| execute regular expression      |                         | regcmp, regex: compile and . . . . .                  | regcmp(S)         |
| make:                           | maintains, updates, and | regenerates groups of programs . . . . .              | make(CP)          |
| regular expression              | regcmp,                 | regex: compile and execute . . . . .                  | regcmp(S)         |



|                                  |                                            |                |
|----------------------------------|--------------------------------------------|----------------|
| executes regular expressions     | regex, regcmp: compiles and                | regex(S)       |
| compile and match routines       | regexp: regular expression                 | regexp(S)      |
| execseg: makes a data            | region executable                          | execseg(S)     |
| locking: locks or unlocks a file | region for reading or writing              | locking(S)     |
| match routines                   | regex: compile and execute                 | regexp(S)      |
| regex: compile and execute       | regular expression regcmp,                 | regcmp(S)      |
| regcmp: compiles                 | regular expressions                        | regcmp(CP)     |
| regcmp: compiles and executes    | regular expressions regex,                 | regex(S)       |
| math package _fpreset:           | reinitializes floating-point               | _fpreset(DOS)  |
| sorted files                     | rejects lines common to two                | comm(C)        |
| comm: Selects or                 | related miscellaneous features/            | Intro(HW)      |
| intro: introduction to machine   | relation for an object library             | lorder(CP)     |
| lorder: finds ordering           | relations                                  | join(C)        |
| join: joins two                  | release indication /acknowledge            | t_rcvrel(NSL)  |
| receipt of an orderly            | release                                    | t_sndrel(NSL)  |
| t_sndrel: initiate an orderly    | brlrel: releases a block buffer            | brlrel(K)      |
| allocated with/                  | sptfree: releases memory previously        | sptfree(K)     |
| DMA channel dma_relse:           | releases previously allocated              | dma_relse(K)   |
| for a common object file         | reloc: relocation information              | reloc(F)       |
| modules 86rel: intel 8086        | relocatable Format for Object              | 86rel(F)       |
| strip: removes symbols and       | relocation bits                            | strip(CP)      |
| strip: removes symbols and       | relocation bits                            | strip(XNX)     |
| of a/ ldrseek, ldnrseek: seek to | relocation entries of a section            | ldrseek(S)     |
| common object file               | relocation information for a               | reloc(F)       |
| reloc:                           | relogin: rename login entry to             | relogin(ADM)   |
| show current layer               | remainder functions /absolute              | floor(S)       |
| value, floor, ceiling and        | reminder service                           | calendar(C)    |
| calendar: invokes a              | remote: executes commands on a             | remote(C)      |
| remote UNIX system               | remote mail received via UUCP              | rmail(ADM)     |
| rmail: submit                    | remote system with debugging on            | uutry(ADM)     |
| uutry: try to contact            | remote terminal                            | ct(C)          |
| ct: spawn getty to a             | remote UNIX system                         | remote(C)      |
| remote: executes commands on a   | remote UNIX                                | uux(C)         |
| uux: executes command on         | rmdir: remove a directory                  | rmdir(S)       |
| rmdir:                           | remove: deletes a file                     | remove(DOS)    |
| file rmb:                        | remove extra blank lines from a            | rmb(M)         |
| removepkg:                       | remove installed package                   | removepkg(ADM) |
| remove: removes filename         | remove: removes temporary files in         | remove(S)      |
| directories specified            | cleantmp: remove temporary files in        | cleantmp(ADM)  |
| package                          | removepkg: remove installed                | removepkg(ADM) |
| file rmdel:                      | removes a delta from an SCCS               | rmdel(CP)      |
| semaphore set or shared/         | ipcrm: removes a message queue,            | ipcrm(ADM)     |
| ipcrm:                           | rmdir: removes directories                 | rmdir(C)       |
| rmdir:                           | unlink: removes directory entry            | unlink(S)      |
| unlink:                          | removes directory names from               | basename(C)    |
| pathnames basename:              | removes filename                           | remove(S)      |
| remove:                          | rm, rmdir: removes files or directories    | rm(C)          |
| rm, rmdir:                       | bits strip: removes symbols and relocation | strip(CP)      |
| bits strip:                      | bits strip: removes symbols and relocation | strip(XNX)     |
| bits strip:                      | rename: changes filename                   | rename(S)      |
| current layer                    | relogin: rename login entry to show        | relogin(ADM)   |

|                                 |                                     |                   |
|---------------------------------|-------------------------------------|-------------------|
| directory                       | rename: renames a file or           | rename(DOS)       |
| rename:                         | renames a file or directory         | rename(DOS)       |
| mv: moves or                    | renames files and directories       | mv(C)             |
| fsck: checks and                | repairs filesystems                 | fsck(ADM)         |
| uniq: reports                   | repeated lines in a file            | uniq(C)           |
| yes: prints string              | repeatedly                          | yes(C)            |
| repinsd, repoutsb, repoutsw,/   | repins: repinsb, repinsw,           | repins(K)         |
| repoutsb, repoutsw,/ repins:    | repinsb, repinsw, repinsd,          | repins(K)         |
| repins: repinsb, repinsw,       | repinsd, repoutsb, repoutsw,/       | repins(K)         |
| repoutsw,/ repins: repinsb,     | repinsw, repinsd, repoutsb,         | repins(K)         |
| fsstat:                         | report filesystem status            | fsstat(ADM)       |
| checkqueue: MMDF queue status   | report generator                    | checkqueue(ADM)   |
| blocks df:                      | report number of free disk          | df(C)             |
| sa2, sadc - system activity     | report package sar: sar, sa1,       | sar(ADM)          |
| activity timex: time a command; | report process data and system      | timex(ADM)        |
| clock:                          | reports CPU time used               | clock(S)          |
| cmchk:                          | reports hard disk block size        | cmchk(C)          |
| ps:                             | reports process status              | ps(C)             |
| file uniq:                      | reports repeated lines in a         | uniq(C)           |
| pstat:                          | reports system information          | pstat(C)          |
| inter-process/ ipc:             | reports the status of               | ipcs(ADM)         |
| vmstat:                         | reports virtual memory statistics   | vmstat(C)         |
| stream fseek, ftell, rewind:    | repositions a file pointer in a     | fseek(S)          |
| /repinsb, repinsw, repinsd,     | repoutsb, repoutsw, repoutsd:/      | repins(K)         |
| /repinsd, repoutsb, repoutsw,   | repoutsd: reads and writes/         | repins(K)         |
| /repinsw, repinsd, repoutsb,    | repoutsw, repoutsd: reads and/      | repins(K)         |
| dma_start: queues DMA           | request                             | dma_start(K)      |
| dma_breakup: sizes DMA          | request into 512-byte blocks        | dma_breakup(K)    |
| t_accept: accept a connect      | request                             | t_accept(NSL)     |
| t_listen: listen for a connect  | request                             | t_listen(NSL)     |
| disksort: adds a block I/O      | request to a device's queue         | disksort(K)       |
| starts/stops the lineprinter    | request /lpshut, lpmove:            | lp sched(ADM)     |
| lp, cancel: Send/cancel         | requests to lineprinter             | lp(C)             |
| breaks up programmed I/O        | requests pio_breakup:               | pio_breakup(K)    |
| not transferred during a DMA    | request /the number of bytes        | dma_resid(K)      |
| the confirmation from a connect | request t_rcvconnect: receive       | t_rcvconnect(NSL) |
| send user-initiated disconnect  | request t_snddis:                   | t_snddis(NSL)     |
| terminal jterm:                 | reset layer of windowing            | jterm(C)          |
| /Awaits and checks access to a  | resource governed by a/             | waitsem(S)        |
| ev_resume:                      | restart a suspended queue           | ev_resume(S)      |
| incremental file/ restore,      | restor: invokes                     | restore(ADM)      |
| incremental filesystem backup/  | restore: AT&T UNIX                  | restore(ADM)      |
| invokes incremental filesystem/ | restore, restor:                    | restore(ADM)      |
| incremental filesystem backup   | restore /AT&T UNIX                  | restore(ADM)      |
| incremental filesystem          | restorer /Invokes XENIX             | xrestore(ADM)     |
| invokes incremental filesystem  | restorer /restor:                   | restore(ADM)      |
| auths: print and/or             | restrict authorizations temporarily | auths(C)          |
| a program promain:              | restrict the execution domain of    | promain(M)        |
| interpreter) rsh: invokes a     | restricted shell (command           | rsh(C)            |
| red: invokes a                  | restricted version of               | ed(C)             |
| disconnect t_rcvdis:            | retrieve information from           | t_rcvdis(NSL)     |



|                                 |                           |                                  |                                  |             |
|---------------------------------|---------------------------|----------------------------------|----------------------------------|-------------|
| object file symbol/             | ldgetname:                | retrieve symbol name for common  | ldgetname(S)                     |             |
|                                 | fp_off, fp_seg:           | return offset and segment        | fp_seg(DOS)                      |             |
| block                           | _msize, _fmsize, _nmsize: | return size of allocated memory  | _msize(DOS)                      |             |
| authentication database fields: |                           | return status based on fields of | fields(S)                        |             |
|                                 | ev_getemask:              | return the current event mask    | ev_getemask(S)                   |             |
|                                 | ev_getemask:              | return the current event mask    | ev_gtemask(S)                    |             |
| system clock in ticks/          | gethz:                    | return the frequency of the      | gethz(S)                         |             |
|                                 | ismpx:                    | return windowing terminal state  | ismpx(C)                         |             |
|                                 | stat: data                | returned by stat system call     | stat(F)                          |             |
|                                 | inp:                      | returns a byte                   | inp(DOS)                         |             |
| console buffer                  | ungetch:                  | returns a character to the       | ungetch(DOS)                     |             |
|                                 | number                    | getpid:                          | returns a process identification | getpid(DOS) |
|                                 | value                     | abs:                             | returns an integer absolute      | abs(S)      |
|                                 | filelength:               | returns length of target file    | filelength(DOS)                  |             |
| number,/                        | major, makedev, minor:    | returns major, new device        | major(K)                         |             |
|                                 | idcheck:                  | returns selected information     | idcheck(ADM)                     |             |
|                                 | _memavl:                  | returns size of available memory | _memavl(DOS)                     |             |
| space                           | stackavail:               | returns size of available stack  | stackavail(DOS)                  |             |
| long integer                    | labs:                     | returns the absolute value of a  | labs(DOS)                        |             |
| heap/                           | _fheapwalk, _nheapwalk:   | returns the address of the next  | _fheapwalk(DOS)                  |             |
|                                 | strlen:                   | returns the length of a string   | strlen(DOS)                      |             |
| transferred during a/           | dma_resid:                | returns the number of bytes not  | dma_resid(K)                     |             |
| currently in the/               | ev_count:                 | returns the number of events     | ev_count(S)                      |             |
| to block data                   | paddr:                    | returns virtual address pointer  | paddr(K)                         |             |
|                                 | value                     | false:                           | returns with a nonzero exit      | false(C)    |
|                                 | true:                     | returns with a zero exit value   | true(C)                          |             |
|                                 | col: filters              | reverse linefeeds                | col(C)                           |             |
|                                 | in a string               | strev:                           | reverses the order of characters | strev(DOS)  |
| pointer in a/                   | fseek, ftell,             | rewind: repositions a file       | fseek(S)                         |             |
| creat: creates a new file or    |                           | rewrites an existing one         | creat(S)                         |             |
| directories                     | rm, rmdir:                | removes files or                 | rm(C)                            |             |
| received via UUCP               | rmail:                    | submit remote mail               | rmail(ADM)                       |             |
| from a file                     | rmb:                      | remove extra blank lines         | rmb(M)                           |             |
| SCCS file                       | rmdel:                    | removes a delta from an          | rmdel(CP)                        |             |
|                                 | rmdir:                    | deletes a directory              | rmdir(DOS)                       |             |
|                                 | rmdir:                    | remove a directory               | rmdir(S)                         |             |
|                                 | rmdir:                    | removes directories              | rmdir(C)                         |             |
| directories                     | rm,                       | rmdir: removes files or          | rm(C)                            |             |
| temporary files in the current/ | rmtmp:                    | closes and deletes all           | rmtmp(DOS)                       |             |
| chroot: changes the             | root                      | directory                        | chroot(S)                        |             |
| chroot: changes                 | root                      | directory for command            | chroot(ADM)                      |             |
| logarithm, power, square        | root                      | functions /exponential,          | exp(S)                           |             |
| supported network               | mmdf:                     | routes mail locally and over any | mmdf(ADM)                        |             |
| error message pointer from last | routine                   | call error /gets                 | strerror(DOS)                    |             |
| error message pointer from last | routine                   | call error /gets                 | strerror(S)                      |             |
| /system services, library       |                           | routines and error numbers       | Intro(S)                         |             |
| field: FIELD library            | routines                  |                                  | field(S)                         |             |
| fieldtype: FIELDTYPE library    | routines                  |                                  | fieldtype(S)                     |             |
| points in a driver object/      | routines:                 | finds driver entry               | routines(ADM)                    |             |
| subsystems : manipulation       | routines                  | for Subsystems database          | subsystems(S)                    |             |
| form: FORM library              | routines                  |                                  | form(S)                          |             |



|                                  |                                  |               |
|----------------------------------|----------------------------------|---------------|
| item: CRT item                   | routines                         | item(S)       |
| ldfcn: common object file access | routines                         | ldfcn(F)      |
| menu: CRT menu                   | routines                         | menu(S)       |
| /selfailure, selwakeup: kernel   | routines supporting select(S)    | select(K)     |
| expression compile and match     | routines regexp: regular         | regexp(S)     |
| scsi_stol, scsi_swap4: SCSI      | routines /scsi_s3tol, scsi_stok, | scsi(K)       |
| ttyflush, ttywait: tty driver    | routines /ttwrite, ttxput,       | tty(K)        |
| virtual address space memory     | routines /vasmapped, vasunbind:  | vas(K)        |
| schedules a time to execute a    | routine timeout, untimeout:      | timeout(K)    |
| /converts a XENIX-style Micnet   | routing file to MMDF/            | mnlist(ADM)   |
| /a XENIX-style UUCP              | routing file to MMDF/            | uulist(ADM)   |
| /hashed database of alias and    | routing information              | dbmbuild(ADM) |
| (command interpreter)            | rsh: invokes a restricted shell  | rsh(C)        |
|                                  | rtc: real time clock interface   | rtc(HW)       |
| multiuser environment rc2:       | run commands performed for       | rc2(ADM)      |
| the operating system rc0:        | run commands performed to stop   | rc0(ADM)      |
| runacct:                         | run daily accounting             | runacct(ADM)  |
|                                  | runacct: run daily accounting    | runacct(ADM)  |
| /prctmp, prdaily, prtacct,       | runacct, shutacct, startup,/     | acctsh(ADM)   |
| priority nice:                   | runs a command at a different    | nice(C)       |
| and quits nohup:                 | runs a command immune to hangups | nohup(C)      |
| mkfifo: make a                   | FIFO special file                | mkfifo(S)     |
| panel:                           | PANEL library routines           | panel(S)      |
| what: identify                   | SCCS files                       | what(CP)      |
| multiplexing poll:               | STREAMS input/output             | poll(S)       |
| activity report/ sar: sar,       | sa1, sa2, sadc - system          | sar(ADM)      |
| report package sar: sar, sa1,    | sa2, sadc - system activity      | sar(ADM)      |
| editing activity                 | sact: prints current SCCS file   | sact(CP)      |
| package sar: sar, sa1, sa2,      | sadc - system activity report    | sar(ADM)      |
|                                  | sag: system activity graph       | sag(ADM)      |
| purge: the policy file of the    | sanitization utility purge(C)    | purge(F)      |
| activity report package sar:     | sar, sa1, sa2, sadc - system     | sar(ADM)      |
| system activity report package   | sar: sar, sa1, sa2, sadc -       | sar(ADM)      |
| fstat:                           | saves file-status information    | fstat(DOS)    |
| space allocation                 | sbrk, brk: changes data segment  | sbrk(S)       |
| allocation brk,                  | sbrk: change data segment space  | brk(S)        |
| work uucico:                     | scan the spool directory for     | uucico(C)     |
| and formats input                | scanf, fscanf, sscanf: converts  | scanf(S)      |
| bfs:                             | scans big files                  | bfs(C)        |
| creates bad track/ badtrk:       | scans fixed disk for flaws and   | badtrk(ADM)   |
| help: asks for help about        | SCCS commands                    | help(CP)      |
| the delta commentary of an       | SCCS delta cdc: changes          | cdc(CP)       |
| comb: combines                   | SCCS deltas                      | comb(CP)      |
| sact: prints current             | SCCS file editing activity       | sact(CP)      |
| prs: prints an                   | SCCS file                        | prs(CP)       |
| rmddel: removes a delta from an  | SCCS file                        | rmddel(CP)    |
| scsfile: format of an            | SCCS file                        | scsfile(F)    |
| val: validates an                | SCCS file                        | val(CP)       |
| makes a delta (change) to an     | SCCS file delta:                 | delta(CP)     |
| admin: creates and administers   | SCCS files                       | admin(CP)     |
| compares two versions of an      | SCCS file sccsdiff:              | sccsdiff(CP)  |

|                                   |                                  |        |                   |
|-----------------------------------|----------------------------------|--------|-------------------|
| undoes a previous get of an       | SCCS file                        | unget: | unget(CP)         |
| of an SCCS file                   | scscdiff: compares two versions  |        | scscdiff(CP)      |
| file                              | scscfile: format of an SCCS      |        | scscfile(F)       |
| system backups                    | schedule: database for automated |        | schedule(ADM)     |
| transport program                 | scheduler for the uucp file      |        | uusched(ADM)      |
| routine                           | schedules a time to execute a    |        | timeout(K)        |
| timeout,untimeout:                | scnhdr: section header for a     |        | scnhdr(F)         |
| common object file                | scr_dump: format of curses       |        | scr_dump(F)       |
| screen image file                 | screen and cursor functions      |        | curses(S)         |
| curses: performs                  | screen                           |        | clear(C)          |
| clear: clears a terminal          | screen color                     |        | setcolor(C)       |
| setcolor: Set                     | screen image file                |        | scr_dump(F)       |
| scr_dump: format of curses        | screen mapping /mapstr,          |        | mapkey(M)         |
| convkey: configure monitor        | screen: tty [01-n],              |        | screen(HW)        |
| color, monochrome, ega,           | screen-oriented display editor   |        | vi(C)             |
| vi, view, vedit: invokes a        | script                           |        | install(M)        |
| install: installation shell       | script xinstall:                 |        | xinstall(ADM)     |
| XENIX installation shell          | SCSI routines /scsi_stok,        |        | scsi(K)           |
| scsi_stol, scsi_swap4:            | scsi: scsi_get_gen_cmd,          |        | scsi(K)           |
| scsi_getdev, scsi_mkadr3,/        | scsi: Small computer systems     |        | scsi(HW)          |
| interface                         | scsi_getdev, scsi_mkadr3,/       |        | scsi(K)           |
| scsi: scsi_get_gen_cmd,           | scsi_get_gen_cmd, scsi_getdev,   |        | scsi(K)           |
| scsi_mkadr3, scsi_s2tos,/ scsi:   | scsi_mkadr3, scsi_s2tos,/ scsi:  |        | scsi(K)           |
| scsi_get_gen_cmd, scsi_getdev,    | scsi_s2tos, scsi_s3tol,/         |        | scsi(K)           |
| /scsi_getdev, scsi_mkadr3,        | scsi_s3tol, scsi_stok,/          |        | scsi(K)           |
| /scsi_mkadr3, scsi_s2tos,         | scsi_stok, scsi_stol,/           |        | scsi(K)           |
| /scsi_s2tos, scsi_s3tol,          | scsi_stol, scsi_swap4: SCSI      |        | scsi(K)           |
| routines /scsi_s3tol, scsi_stok,  | scsi_swap4: SCSI routines        |        | scsi(K)           |
| /scsi_stok, scsi_stol,            | sdb: invokes symbolic debugger   |        | sdb(CP)           |
|                                   | sddate: prints and sets backup   |        | sddate(C)         |
| dates                             | sdenter, sdleave: Synchronizes   |        | sdenter(S)        |
| access to a shared data/          | sdevice file /of vectors         |        | vectorsinuse(ADM) |
| currently specified in the        | sdevice: local device            |        | sdevice(F)        |
| configuration file                | sdfree: attaches and detaches a  |        | sdget(S)          |
| shared data segment sdget,        | sdget, sdfree: attaches and      |        | sdget(S)          |
| detaches a shared data segment    | sdgetv, sdwaitv: Synchronizes    |        | sdgetv(S)         |
| shared data access                | sdiff: compares files            |        | sdiff(C)          |
| side-by-side                      | sdleave: Synchronizes access to  |        | sdenter(S)        |
| a shared data segment sdenter,    | sdwaitv: Synchronizes shared     |        | sdgetv(S)         |
| data access sdgetv,               | search and update                |        | lsearch(S)        |
| lsearch, lfind: performs linear   | search                           |        | bsearch(S)        |
| bsearch: performs a binary        | search tables hsearch,           |        | hsearch(S)        |
| hcreate, hdestroy: manages hash   | search trees tsearch, tfind,     |        | tsearch(S)        |
| tdelete, twalk: manages binary    | searches a file for a pattern    |        | grep(C)           |
| grep, egrep, fgrep:               | searches for and prints process  |        | acctcom(ADM)      |
| accounting files acctcom:         | searches for and processes a     |        | awk(C)            |
| pattern in a file awk:            | search lfind,                    |        | lfind(DOS)        |
| lsearch: performs a linear array  | section header for a common      |        | scnhdr(F)         |
| object file scnhdr:               | section header of a common       |        | ldshread(S)       |
| object/ /read an indexed/named    | section of a common object file  |        | ldlseek(S)        |
| /seek to line number entries of a | section of a common object file  |        | ldrseek(S)        |
| /seek to relocation entries of a  |                                  |        |                   |



|                                  |                                       |                 |
|----------------------------------|---------------------------------------|-----------------|
| /seek to an indexed/named        | section of a common object file . .   | ldsseek(S)      |
| the object file comment          | section mcs: manipulate . . . .       | mcs(CP)         |
| getty initcond: special          | security actions for init and . . .   | initcond(ADM)   |
| /verify machine is suitable for  | security port . . . . .               | check_data(S)   |
| description subsystem:           | security subsystem component . .      | subsystem(M)    |
|                                  | sed: invokes the stream editor . .    | sed(C)          |
| /getseed, setseed: obtain or set | seed for random number generator .    | seed(S)         |
| or set seed for random number/   | seed: getseed, setseed: obtain . .    | seed(S)         |
| uniformly distributed srand48,   | seed48, lcong48: generates . . .      | drand48(S)      |
| section of a/ ldsseek, ldsseek:  | seek to an indexed/named . . . .      | ldsseek(S)      |
| section of a/ ldlseek, ldlseek:  | seek to line number entries of a . .  | ldlseek(S)      |
| section of a/ ldrseek, ldrseek:  | seek to relocation entries of a . . . | ldrseek(S)      |
| of a common object/ ldohseek:    | seek to the optional file header . .  | ldohseek(S)     |
| common object file ldtbseek:     | seek to the symbol table of a . . .   | ldtbseek(S)     |
| brkctl: allocates data in a far  | segment . . . . .                     | brkctl(S)       |
| FP_SEG: get the offset and       | segment of an address FP_OFF, . .     | fp_off(DOS)     |
| shmget: gets a shared memory     | segment . . . . .                     | shmget(S)       |
| brk, sbrk: change data           | segment space allocation . . . .      | brk(S)          |
| sbrk, brk: changes data          | segment space allocation . . . .      | sbrk(S)         |
| fp_seg: return offset and        | segment fp_off, . . . . .             | fp_seg(DOS)     |
| and detaches a shared data       | segment /sdfree: attaches . . . .     | sdget(S)        |
| access to a shared data          | segment /sdleave: Synchronizes .      | sdenter(S)      |
|                                  | segread: command description . .      | segread(DOS)    |
| selwakeup: kernel routines/      | select: selsuccess, selfailure, . .   | select(K)       |
| multiplexing                     | select: synchronous I/O . . . .       | select(S)       |
| greek:                           | select terminal filter . . . . .      | greek(C)        |
| a file cut: cuts out             | selected fields of each line of . .   | cut(C)          |
| idcheck: returns                 | selected information . . . . .        | idcheck(ADM)    |
| dump: dump                       | selected parts of an object file . .  | dump(CP)        |
| binary files hdr: displays       | selected parts of executable . . .    | hdr(CP)         |
| to two sorted files comm:        | selects or rejects lines common . .   | comm(C)         |
| kernel routines supporting       | select(S) /selwakeup: . . . . .       | select(K)       |
| routines/ select: selsuccess,    | selfailure, selwakeup: kernel . . .   | select(K)       |
| selwakeup: kernel/ select:       | selsuccess, selfailure, . . . . .     | select(K)       |
| select: selsuccess, selfailure,  | selwakeup: kernel routines/ . . .     | select(K)       |
| opensem: opens a                 | semaphore . . . . .                   | opensem(S)      |
| semctl: controls                 | semaphore operations . . . . .        | semctl(S)       |
| semop: performs                  | semaphore operations . . . . .        | semop(S)        |
| ipcrm: removes a message queue,  | semaphore set or shared memory .      | ipcrm(ADM)      |
| to a resource governed by a      | semaphore /and checks access . .      | waitsem(S)      |
| creates an instance of a binary  | semaphore creatsem: . . . . .         | creatsem(S)     |
| files lockf: provide             | semaphores and record locking on .    | lockf(S)        |
| semget: gets set of              | semaphores . . . . .                  | semget(S)       |
| signals a process waiting on a   | semaphore sigsem: . . . . .           | sigsem(S)       |
| operations                       | semctl: controls semaphore . . .      | semctl(S)       |
|                                  | semget: gets set of semaphores . .    | semget(S)       |
| fsphoto: performs periodic       | semi-automated system backups . .     | fsphoto(ADM)    |
| operations                       | semop: performs semaphore . . .       | semop(S)        |
| t_sndudata:                      | send a data unit . . . . .            | t_sndudata(NSL) |
| putmsg:                          | send a message on a stream . . .      | putmsg(S)       |
| hello:                           | send a message to another user . .    | hello(ADM)      |



|                                |             |                                                  |                                     |
|--------------------------------|-------------|--------------------------------------------------|-------------------------------------|
| a connection                   | t_snd:      | send data or expedited data over                 | t_snd(NSL)                          |
| execution program              | raise:      | send signal <i>sig</i> to                        | raise(S)                            |
| request                        | t_snddis:   | send user-initiated disconnect                   | t_snddis(NSL)                       |
| lineprinter                    | lp, cancel: | send/cancel requests to                          | lp(C)                               |
| group of processes             | kill:       | sends a signal to a process or a                 | kill(S)                             |
|                                | signal:     | sends a signal to a process                      | signal(K)                           |
| program                        | raise:      | sends a signal to the executing                  | raise(DOS)                          |
| mail                           | mail:       | sends, reads or disposes of                      | mail(C)                             |
|                                | psignal:    | sends signal to a process                        | psignal(K)                          |
| /sys_errlist, sys_nerr, errno: |             | sends system error messages                      | perror(S)                           |
| receive data or expedited data |             | sent over a connection                           | t_rcv(NSL)                          |
| mesg:                          |             | permits or denies messages                       | mesg(C)                             |
| file to printer attached to a  |             | serial console                                   | /Print consoleprint(ADM)            |
|                                | mscreen:    | serial multiscreens utility                      | mscreen(M)                          |
| , tty2[A-H]:                   |             | interface to serial ports                        | /, tty2[a-h] serial(HW)             |
| calendar:                      |             | invokes a reminder service                       | calendar(C)                         |
| error/ intro:                  |             | introduces system services, library routines and | Intro(S)                            |
| introduction to the Network    |             | Services library intro:                          | intro(NSL)                          |
|                                | setsid:     | create session and set process ID                | setsid(S)                           |
| /open and access audit         |             | session data on a record basis                   | audit(S)                            |
| disable auditing for the next  |             | session                                          | chg_audit: enables and              |
| map of the ASCII character     |             | set ascii:                                       | ascii(M)                            |
| buffering to a stream          |             | setbuf, setvbuf:                                 | assigns setbuf(S)                   |
| real-time (time of day) clock  |             | setclock:                                        | Sets the system setclock(ADM)       |
|                                |             | setcolor:                                        | Set screen color setcolor(C)        |
| getdvagent, getdvnagm,         |             | setdvagent, enddvagent,/                         | getdvagent(S)                       |
| error code                     |             | seterror:                                        | sets u.u_error with seterror(K)     |
| setuid,                        |             | setgid:                                          | Sets user and group IDs setuid(S)   |
| getgrent, getgrgid, getgnam,   |             | setgrent, endgrent:                              | get group/ getgrent(S)              |
|                                |             | setgroups:                                       | set group access list setgroups(S)  |
| nonlocal "goto"                |             | setjmp, longjmp:                                 | performs a setjmp(S)                |
| keys                           |             | setkey:                                          | assigns the function setkey(C)      |
| international environment      |             | setlocale:                                       | Set or read setlocale(S)            |
|                                |             | setluid:                                         | set login user ID setluid(S)        |
| table                          |             | setmnt:                                          | establishes /etc/mnttab setmnt(ADM) |
|                                |             | setmode:                                         | Sets translation mode setmode(DOS)  |
| for job control                |             | setpgid:                                         | set process group ID setpgid(S)     |
|                                |             | setpgrp:                                         | Sets process group ID setpgrp(S)    |
| getprcment, getprcmnam,        |             | setprcment, endprcment,/                         | getprcment(S)                       |
| getprdfent, getprdfnam,        |             | setprdfent, endprdfent,/                         | getprdfent(S)                       |
| getprfient, getprfinam,        |             | setprfient, endprfient,/                         | getprfient(S)                       |
| for this process               |             | setpriv:                                         | set system privileges setpriv(S)    |
| /getprpwuid, getprpwnam,       |             | setprpwent, endprpwent,/                         | getprpwent(S)                       |
| getprtcent, getprtcnam,        |             | setprtcent, endprtcent,/                         | getprtcent(S)                       |
| getpwent, getpwuid, getpwnam,  |             | setpwent, endpwent:                              | gets/ getpwent(S)                   |
| alarm:                         |             | sets a process' alarm clock                      | alarm(S)                            |
| to one charater                | strset:     | sets all characters in a string                  | strset(DOS)                         |
| mask                           | umask:      | sets and gets file creation                      | umask(S)                            |
| sddate:                        |             | prints and sets backup dates                     | sddate(C)                           |
| execution env:                 |             | sets environment for command                     | env(C)                              |
| ev_setemask:                   |             | sets event mask                                  | ev_setemask(S)                      |

|                                 |                                            |                 |
|---------------------------------|--------------------------------------------|-----------------|
| ev_setmask:                     | sets event mask . . . . .                  | ev_setmask(S)   |
| modification times utime:       | sets file access and . . . . .             | utime(S)        |
| utime:                          | sets file modification time . . . . .      | utime(DOS)      |
| umask:                          | sets file-creation mode mask . . . . .     | umask(C)        |
| word _control87: gets and       | sets floating-point control . . . . .      | _control87(DOS) |
| (zero) bzero:                   | sets memory locations to 0 . . . . .       | bzero(K)        |
| setpgrp:                        | sets process group ID . . . . .            | setpgrp(S)      |
| for a portion of a/ locking:    | sets read and write permissions . . . . .  | locking(DOS)    |
| sigset: manipulate signal       | sets . . . . .                             | sigset(S)       |
| tset:                           | sets terminal modes . . . . .              | tset(C)         |
| speed, and line/ getty:         | sets terminal type, modes, . . . . .       | getty(M)        |
| base cmos: displays and         | sets the configuration data . . . . .      | cmos(HW)        |
| date: prints and                | sets the date . . . . .                    | date(C)         |
| umask:                          | sets the file permission mask . . . . .    | umask(DOS)      |
| for a stream fsetpos:           | sets the file position indicator . . . . . | fsetpos(DOS)    |
| for a stream fsetpos:           | sets the file position indicator . . . . . | fsetpos(S)      |
| a video device vidi:            | sets the font and video mode for . . . . . | vidi(C)         |
| stty:                           | sets the options for a terminal . . . . .  | stty(C)         |
| of day) clock setclock:         | sets the system real-time (time . . . . .  | setclock(ADM)   |
| stime:                          | sets the time . . . . .                    | stime(S)        |
| setmode:                        | sets translation mode . . . . .            | setmode(DOS)    |
| trchan: translate character     | sets . . . . .                             | trchan(M)       |
| for DMA transfer dma_param:     | sets up a DMA controller chip . . . . .    | dma_param(K)    |
| time profile:                   | sets up an environment at login . . . . .  | profile(M)      |
| setuid, setgid:                 | sets user and group IDs . . . . .          | setuid(S)       |
| ulimit: gets and                | sets user limits . . . . .                 | ulimit(S)       |
| seterror:                       | sets u.u_error with error code . . . . .   | seterror(K)     |
| random number/ seed: getseed,   | setseed: obtain or set seed for . . . . .  | seed(S)         |
| process ID                      | setuid: create session and set . . . . .   | setuid(S)       |
| modification dates of files     | settime: changes the access and . . . . .  | settime(ADM)    |
| gettydefs: Speed and terminal   | settings used by getty . . . . .           | gettydefs(F)    |
| group IDs                       | setuid, setgid: Sets user and . . . . .    | setuid(S)       |
| stream buffering and size       | setvbuf: allow user control over . . . . . | setvbuf(DOS)    |
| stream setbuf,                  | setvbuf: assigns buffering to a . . . . .  | setbuf(S)       |
| file                            | sfsys: local filesystem type . . . . .     | sfsys(F)        |
| data in a/ sputl,               | sgetl: accesses long integer . . . . .     | sputl(S)        |
| interpreter                     | sh: invokes the shell command . . . . .    | sh(C)           |
| sdgetv, sdwaitv: Synchronizes   | shared data access . . . . .               | sdgetv(S)       |
| sdfree: attaches and detaches a | shared data segment sdget, . . . . .       | sdget(S)        |
| synchronizes access to a        | shared data segment /sdleave: . . . . .    | sdenter(S)      |
| chkshlib: compare               | shared libraries tool . . . . .            | chkshlib(CP)    |
| mkshlib: create a               | shared library . . . . .                   | mkshlib(CP)     |
| shmctl: controls                | shared memory operations . . . . .         | shmctl(S)       |
| shmop: performs                 | shared memory operations . . . . .         | shmop(S)        |
| shmget: gets a                  | shared memory segment . . . . .            | shmget(S)       |
| message queue, semaphore set or | shared memory ipcrm: removes a . . . . .   | ipcrm(ADM)      |
| sopen: opens a file for         | shared reading and writing . . . . .       | sopen(DOS)      |
| rsh: invokes a restricted       | shell (command interpreter) . . . . .      | rsh(C)          |
| sh: invokes the                 | shell command interpreter . . . . .        | sh(C)           |
| c-like syntax csh: invokes a    | shell command interpreter with . . . . .   | csh(C)          |
| system: executes a              | shell command . . . . .                    | system(S)       |



|                                  |                                        |                |
|----------------------------------|----------------------------------------|----------------|
|                                  | shl: shell layer manager . . . . .     | shl(C)         |
| change login, group, or dialup   | shell password passwd: . . . . .       | passwd(C)      |
| /shutacct, startup, turnacct -   | shell procedures for accounting . . .  | acctsh(ADM)    |
| /shutacct, startup, turnacct     | shell procedures for/ . . . . .        | acctsh(ADM)    |
| install: installation            | shell script . . . . .                 | install(M)     |
| xinstall: XENIX installation     | shell script . . . . .                 | xinstall(ADM)  |
|                                  | shl: Shell layer manager . . . . .     | shl(C)         |
| operations                       | shmctl: controls shared memory . . .   | shmctl(S)      |
| segment                          | shmget: gets a shared memory . . .     | shmget(S)      |
| operations                       | shmop: performs shared memory . . .    | shmop(S)       |
| nap: Suspends execution for a    | short interval . . . . .               | nap(S)         |
| /prdaily, prtacct, runacct,      | shutacct, startup, turnacct -/ . . .   | acctsh(ADM)    |
| halts the CPU                    | shutdn: flushes block I/O and . . .    | shutdn(S)      |
| processing                       | shutdown: terminates all . . . . .     | shutdown(ADM)  |
| closes out the filesystems and   | shuts down the system /reboot: . . .   | haltsys(ADM)   |
| sdiff: compares files            | side-by-side . . . . .                 | sdiff(C)       |
| signal action                    | sigaction: examine and change . . .    | sigaction(S)   |
| sigsetjmp,                       | siglongjmp: non-local jumps . . .      | sigsetjmp(S)   |
| sigaction: examine and change    | signal action . . . . .                | sigaction(S)   |
| taken upon receipt of a signal   | signal: defines the action to be . . . | signal(DOS)    |
| program raise: send              | signal sig to execution . . . . .      | raise(S)       |
| suspends a process until a       | signal occurs pause: . . . . .         | pause(S)       |
| process                          | signal: sends a signal to a . . . . .  | signal(K)      |
| sigset: manipulate               | signal sets . . . . .                  | sigset(S)      |
| sigsuspend: wait for             | signal . . . . .                       | sigsuspend(S)  |
| upon receipt of a signal         | signal: Specifies what to do . . . .   | signal(S)      |
| of processes kill: Sends a       | signal to a process or a group . . .   | kill(S)        |
| psignal: sends                   | signal to a process . . . . .          | psignal(K)     |
| signal: sends a                  | signal to a process . . . . .          | signal(K)      |
| raise: sends a                   | signal to the executing program . .    | raise(DOS)     |
| to be taken upon receipt of a    | signal /defines the action . . . . .   | signal(DOS)    |
| semaphore sigsem:                | signals a process waiting on a . . .   | sigsem(S)      |
| iodone:                          | signals I/O completion . . . . .       | iodone(K)      |
| sigpending: examine pending      | signals . . . . .                      | sigpending(S)  |
| what to do upon receipt of a     | signal signal: Specifies . . . . .     | signal(S)      |
| examine and change blocked       | signals sigprocmask: . . . . .         | sigprocmask(S) |
| gsignal: implements software     | signals ssignal, . . . . .             | ssignal(S)     |
| signals                          | sigpending: examine pending . . .      | sigpending(S)  |
| blocked signals                  | sigprocmask: examine and change .      | sigprocmask(S) |
| waiting on a semaphore           | sigsem: Signals a process . . . . .    | sigsem(S)      |
|                                  | sigset: manipulate signal sets . . .   | sigset(S)      |
| jumps                            | sigsetjmp, siglongjmp: non-local .     | sigsetjmp(S)   |
|                                  | sigsuspend: wait for signal . . . .    | sigsuspend(S)  |
| atan2: performs trigonometric/   | sin, cos, tan, asin, acos, atan, . . . | trig(S)        |
| sulogin: access                  | single-user mode . . . . .             | sulogin(ADM)   |
| hyperbolic functions             | sinh, cosh, tanh: performs . . . .     | sinh(S)        |
| cmchk: reports hard disk block   | size . . . . .                         | cmchk(C)       |
| chsize: changes the              | size of a file . . . . .               | chsize(S)      |
| memory/ _expand: changes the     | size of a previously allocated . . .   | _expand(DOS)   |
| _msize, _fmsize, _nmsize: return | size of allocated memory block . . .   | _msize(DOS)    |
| size: prints the                 | size of an object file . . . . .       | size(CP)       |



|                                  |                                            |                    |
|----------------------------------|--------------------------------------------|--------------------|
| size: prints the                 | size of an object file . . . . .           | size(XNX)          |
| _memavl: returns                 | size of available memory . . . . .         | _memavl(DOS)       |
| stackavail: returns              | size of available stack space . . . . .    | stackavail(DOS)    |
| jwin: print                      | size of layer . . . . .                    | jwin(C)            |
| object file                      | size: prints the size of an . . . . .      | size(CP)           |
| object file                      | size: prints the size of an . . . . .      | size(XNX)          |
| over stream buffering and        | size /allow user control . . . . .         | setvbuf(DOS)       |
| blocks dma_breakup:              | sizes DMA request into 512-byte . . . . .  | dma_breakup(K)     |
| interval                         | sleep: Suspends execution for an . . . . . | sleep(C)           |
| interval                         | sleep: Suspends execution for an . . . . . | sleep(S)           |
| temporarily                      | sleep: suspends processing . . . . .       | sleep(K)           |
| wakeup: wakes up a               | sleeping process . . . . .                 | wakeup(K)          |
| current/ tty slot: finds the     | slot in the utmp file of the . . . . .     | ttyslot(S)         |
| spline: interpolates             | smooth curve . . . . .                     | spline(C)          |
| swconfig: produces a list of the | software modifications to the/ . . . . .   | swconfig(C)        |
| ssignal, gsignal: implements     | software signals . . . . .                 | ssignal(S)         |
| reading and writing              | sopen: opens a file for shared . . . . .   | sopen(DOS)         |
| qsort: performs a quicker        | sort . . . . .                             | qsort(S)           |
| or rejects lines common to two   | sort: Sorts and merges files . . . . .     | sort(C)            |
| tsort:                           | sorted files comm: Selects . . . . .       | comm(C)            |
| sort:                            | sorts a file topologically . . . . .       | tsort(CP)          |
| object file list: produce C      | sorts and merges files . . . . .           | sort(C)            |
| an error message file from C     | source listing from a common . . . . .     | list(CP)           |
| brk, sbrk: change data segment   | source mkstr: creates . . . . .            | mkstr(CP)          |
| sbrk, brk: changes data segment  | space allocation . . . . .                 | brk(S)             |
| passes character between user    | space allocation . . . . .                 | sbrk(S)            |
| bcopy: copies bytes in kernel    | space and the kernel /passc: . . . . .     | cpass(K)           |
| idspace: investigates free       | space . . . . .                            | bcopy(K)           |
| /vasunbind: virtual address      | space . . . . .                            | idspace(ADM)       |
| bytes between user and kernel    | space memory routines . . . . .            | vas(K)             |
| gets a character from user data  | space copyin, copyout: copies . . . . .    | copyin(K)          |
| one 32-bit word from user data   | space fubyte: . . . . .                    | fubyte(K)          |
| returns size of available stack  | space fuword: gets . . . . .               | fuword(K)          |
| stores a character in user data  | space stackavail: . . . . .                | stackavail(DOS)    |
| a 32-bit word in user data       | space subyte: . . . . .                    | subyte(K)          |
| ct:                              | space suword: stores . . . . .             | suword(K)          |
| process                          | spawn getty to a remote terminal . . . . . | ct(C)              |
| spawnl,                          | spawnl, spawnvp: creates a new . . . . .   | spawn(DOS)         |
| movedata: copies bytes from a    | spawnvp: creates a new process . . . . .   | spawn(DOS)         |
| sysi86: machine                  | specific address . . . . .                 | movedata(DOS)      |
| fspec: format                    | specific functions . . . . .               | sysi86(S)          |
| idmkinit: read files containing  | specification in text files . . . . .      | fspec(F)           |
| purge: overwrites                | specifications . . . . .                   | idmkinit(ADM)      |
| /major device numbers currently  | specified files . . . . .                  | purge(C)           |
| /the list of vectors currently   | specified in the mdevice file . . . . .    | major sinuse(ADM)  |
| delays process execution for     | specified in the sdevice/ . . . . .        | vector sinuse(ADM) |
| cron: executes commands at       | specified time delay: . . . . .            | delay(K)           |
| temporary files in directories   | specified times . . . . .                  | cron(C)            |
| receipt of a signal signal:      | specified cleantmp: remove . . . . .       | cleantmp(ADM)      |
| /Sets terminal type, modes,      | specifies what to do upon . . . . .        | signal(S)          |
|                                  | speed, and line discipline . . . . .       | getty(M)           |

## Permuted Index

|                                  |                                          |                 |
|----------------------------------|------------------------------------------|-----------------|
| /set terminal type, modes,       | speed, and line discipline . . . .       | uugetty(ADM)    |
| by getty gettydefs:              | speed and terminal settings used .       | gettydefs(F)    |
| hashcheck: finds spelling/       | spell, hashmake, spellin, . . . .        | spell(C)        |
| spelling/ spell, hashmake,       | spellin, hashcheck: finds . . . .        | spell(C)        |
| spellin, hashcheck: finds        | spelling errors /hashmake, . . . .       | spell(C)        |
| spl4, spl5, spl6, spl7, splbuf,/ | spl: spl0, spl1, spl2, spl3, . . . .     | spl(K)          |
| spl5, spl6, spl7, splbuf,/ spl:  | spl0, spl1, spl2, spl3, spl4, . . . .    | spl(K)          |
| spl6, spl7, splbuf,/ spl: spl0,  | spl1, spl2, spl3, spl4, spl5, . . . .    | spl(K)          |
| spl7, splbuf,/ spl: spl0, spl1,  | spl2, spl3, spl4, spl5, spl6, . . . .    | spl(K)          |
| splbuf,/ spl: spl0, spl1, spl2,  | spl3, spl4, spl5, spl6, spl7, . . . .    | spl(K)          |
| spl: spl0, spl1, spl2, spl3,     | spl4, spl5, spl6, spl7, splbuf,/ . . . . | spl(K)          |
| spl0, spl1, spl2, spl3, spl4,    | spl5, spl6, spl7, splbuf,/ spl: . . . .  | spl(K)          |
| /spl1, spl2, spl3, spl4, spl5,   | spl6, spl7, splbuf, splcli,/ . . . .     | spl(K)          |
| /spl2, spl3, spl4, spl5, spl6,   | spl7, splbuf, splcli, splhi,/ . . . .    | spl(K)          |
| /spl3, spl4, spl5, spl6, spl7,   | splbuf, splcli, splhi, splni,/ . . . .   | spl(K)          |
| /spl4, spl5, spl6, spl7, splbuf, | splcli, splhi, splni, splpp,/ . . . .    | spl(K)          |
| /spl6, spl7, splbuf, splcli,     | splhi, splni, splpp, spltty,/ . . . .    | spl(K)          |
| curve                            | spline: interpolates smooth . . . .      | spline(C)       |
| pieces                           | split: Splits a file into . . . . .      | split(C)        |
| split:                           | splits a file into pieces . . . . .      | split(C)        |
| context csplit:                  | splits files according to . . . . .      | csplit(C)       |
| into a/ frexp, ldexp, modf:      | splits floating-point number . . . .     | frexp(S)        |
| /spl7, splbuf, splcli, splhi,    | splni, splpp, spltty, splx:/ . . . .     | spl(K)          |
| splbuf, splcli, splhi, splni,    | splpp, spltty, splx:/ /spl7, . . . .     | spl(K)          |
| /splcli, splhi, splni, splpp,    | spltty, splx: block/permit/ . . . .      | spl(K)          |
| /splhi, splni, splpp, spltty,    | splx: block/permit interrupts . . . .    | spl(K)          |
| uclean: uucp                     | spool directory clean-up . . . . .       | uclean(ADM)     |
| uucico: Scan the                 | spool directory for work . . . . .       | uucico(C)       |
| configures the lineprinter       | spooling system lpadmin: . . . . .       | lpadmin(ADM)    |
| printf, fprintf,                 | sprintf: formats output . . . . .        | printf(S)       |
| memory or maps a device into/    | sptalloc: allocates temporary . . . .    | sptalloc(K)     |
| memory previously allocated with | sptalloc sptfree: releases . . . . .     | sptfree(K)      |
| previously allocated with/       | sptfree: releases memory . . . . .       | sptfree(K)      |
| integer data in a/               | sputl, sgetl: accesses long . . . . .    | sputl(S)        |
| exponential,/ exp, log, pow,     | sqr, log10: performs . . . . .           | exp(S)          |
| exponential, logarithm, power,   | square root functions /Performs . . . .  | exp(S)          |
| number rand,                     | srand: generates a random . . . . .      | rand(S)         |
| generates uniformly/             | rand48, seed48, lcong48: . . . . .       | drand48(S)      |
| input scanf, fscanf,             | sscanf: converts and formats . . . .     | scanf(S)        |
| software signals                 | ssignal, gsignal: implements . . . .     | ssignal(S)      |
| returns size of available        | stack space stackavail: . . . . .        | stackavail(DOS) |
| bytes from the program's         | stack alloca: allocates . . . . .        | alloca(DOS)     |
| available stack space            | stackavail: returns size of . . . . .    | stackavail(DOS) |
| output stdio: performs           | standard buffered input and . . . .      | stdio(S)        |
| converts Rational FORTRAN into   | standard FORTRAN ratfor: . . . . .       | ratfor(CP)      |
| gets: gets a string from the     | standard input . . . . .                 | gets(CP)        |
| communication package ftok:      | standard interprocess . . . . .          | stdipc(S)       |
| pr: prints files on the          | standard output . . . . .                | pr(C)           |
| check uids or gids from program  | start identity: get or . . . . .         | identity(S)     |
| lpsched, lpshut, lpmove:         | starts/stops the lineprinter/ . . . .    | lpsched(ADM)    |
| /prtacct, runacct, shutacct,     | startup, turnacct - shell/ . . . . .     | acctsh(ADM)     |



|                                  |                                          |                 |
|----------------------------------|------------------------------------------|-----------------|
| system call                      | stat: data returned by stat . . . .      | stat(F)         |
|                                  | stat, fstat: gets file status . . . .    | stat(S)         |
|                                  | stat: gets file status . . . . .         | stat(DOS)       |
| stat: data returned by           | stat system call . . . . .               | stat(F)         |
| information                      | statfs: get filesystem . . . . .         | statfs(S)       |
| ustat: gets filesystem           | statistics . . . . .                     | ustat(S)        |
| virtual memory                   | statistics. vmstat: reports . . . .      | vmstat(C)       |
| xts: extract and print xt driver | statistics . . . . .                     | xts(ADM)        |
| authentication/ fields: return   | status based on fields of . . . . .      | fields(S)       |
| fsstat: report filesystem        | status . . . . .                         | fsstat(ADM)     |
| lpstat: prints lineprinter       | status information . . . . .             | lpstat(C)       |
| uustat: uucp                     | status inquiry and job control . . . .   | uustat(C)       |
| communication/ ipc: reports the  | status of inter-process . . . . .        | ipc(S)          |
| ps: reports process              | status . . . . .                         | ps(C)           |
| checkqueue: MMDF queue           | status report generator . . . . .        | checkqueue(ADM) |
| stat: gets file                  | status . . . . .                         | stat(DOS)       |
| stat, fstat: gets file           | status . . . . .                         | stat(S)         |
| _status87: gets floating-point   | status word . . . . .                    | _status87(DOS)  |
| and clears the floating-point    | status word _clear87: gets . . . .       | _clear87(DOS)   |
| status word                      | _status87: gets floating-point . . . .   | _status87(DOS)  |
| fileno: determines stream        | status error, feof, clearerr, . . . .    | error(S)        |
| buffered input and output        | stdio: performs standard . . . . .       | stdio(S)        |
|                                  | stime: Sets the time . . . . .           | stime(S)        |
| stopio:                          | stop further I/O to an open file . . . . | stopio(S)       |
| waits for a child process to     | stop or terminate wait: . . . . .        | wait(S)         |
| rc0: run commands performed to   | stop the operating system . . . . .      | rc0(ADM)        |
| open file                        | stopio: stop further I/O to an . . . .   | stopio(S)       |
| compress: compress data for      | storage . . . . .                        | compress(C)     |
| nextkey:/ dbm: fetch,            | store, delete, firstkey, . . . . .       | dbm(S)          |
| uncompress: uncompress a         | stored file . . . . .                    | compress(C)     |
| zcat: display a                  | stored file . . . . .                    | compress(C)     |
| data space suword:               | stores a 32-bit word in user . . . .     | suword(K)       |
| space subyte:                    | stores a character in user data . . . .  | subyte(K)       |
| information dosxterr: gets and   | stores extended error . . . . .          | dosxterr(DOS)   |
| stream's file/ fgetpos: gets and | stores the current value of a . . . .    | fgetpos(DOS)    |
| stream's file/ fgetpos: gets and | stores the current value of a . . . .    | fgetpos(S)      |
| queue: MMDF queue files for      | storing mail in transit . . . . .        | queue(ADM)      |
| trace messages                   | strace: prints STREAMS . . . . .         | strace(ADM)     |
| strcsn, strdup, stricmp:/        | strcat, strchr, strncmp, strcpy, . . . . | strcat(DOS)     |
| strcsn, strdup,/ strcat,         | strchr, strncmp, strcpy, . . . . .       | strcat(DOS)     |
| logger cleanup program           | strclean: STREAMS error . . . . .        | strclean(ADM)   |
| strdup,/ strcat, strchr,         | strcmp, strcpy, strcsn, . . . . .        | strcat(DOS)     |
| collation of/ strxfrm, strnxfm,  | strcoll, strncoll: handles . . . . .     | collation(S)    |
| strcat, strchr, strncmp,         | strcpy, strcsn, strdup,/ . . . . .       | strcat(DOS)     |
| strcat, strchr, strncmp, strcpy, | strcsn, strdup, stricmp:/ . . . . .      | strcat(DOS)     |
| operations                       | strdup: performs string . . . . .        | string(S)       |
| /strncmp, strcpy, strcsn,        | strdup, stricmp: perform/ . . . . .      | strcat(DOS)     |
| setvbuf: allow user control over | stream buffering and size . . . . .      | setvbuf(DOS)    |
| invokes the                      | stream editor. sed: . . . . .            | sed(C)          |
| fopen, freopen, fdopen: opens a  | stream . . . . .                         | fopen(S)        |
| fread: reads from the input      | stream . . . . .                         | fread(DOS)      |



|                                  |                                        |               |
|----------------------------------|----------------------------------------|---------------|
| freopen: assigns a new file to a | stream . . . . .                       | freopen(DOS)  |
| fwrite: writes to the output     | stream . . . . .                       | fwrite(DOS)   |
| getmsg: get next message off a   | stream . . . . .                       | getmsg(S)     |
| putmsg: send a message on a      | stream . . . . .                       | putmsg(S)     |
| puts, fputs: puts a string on a  | stream . . . . .                       | puts(S)       |
| clearerr, fileno: determines     | stream status  ferror, feof, . . . .   | ferror(S)     |
| fflush: closes or flushes a      | stream  fclose, . . . . .              | fclose(S)     |
| gets a character from a          | stream  fgetc, fgetchar: . . . . .     | fgetc(DOS)    |
| fputc: write a character to a    | stream  fputc, . . . . .               | fputc(DOS)    |
| repositions a file pointer in a  | stream  fseek, ftell, rewind: . . . .  | fseek(S)      |
| file position indicator for a    | stream  fsetpos: sets the . . . . .    | fsetpos(DOS)  |
| file position indicator for a    | stream  fsetpos: sets the . . . . .    | fsetpos(S)    |
| gets character or word from a    | stream  /fgetchar, fgetc, getw: . . .  | getc(S)       |
| fgets: gets a string from a      | stream  gets, . . . . .                | gets(S)       |
| prints the first few lines of a  | stream  head: . . . . .                | head(C)       |
| puts a character or word on a    | stream  /putchar, fputc, putw: . . .   | putc(S)       |
| fclose, fcloseall: closes        | streams . . . . .                      | fclose(DOS)   |
| /stores the current value of a   | stream's file position indicator . . . | fgetpos(DOS)  |
| /stores the current value of a   | stream's file position indicator . . . | fgetpos(S)    |
| /repoutsd: reads and writes      | streams of device data . . . . .       | repins(K)     |
| setvbuf: assigns buffering to a  | stream  setbuf, . . . . .              | setbuf(S)     |
| cleanup program  strclean:       | STREAMS error logger . . . . .         | strclean(ADM) |
| daemon  strerr:                  | STREAMS error logger . . . . .         | strerr(ADM)   |
| strace: prints                   | STREAMS trace messages . . . . .       | strace(ADM)   |
| pushes character back into input | stream  ungetc: . . . . .              | ungetc(S)     |
| logger  daemon                   | strerr: STREAMS error . . . . .        | strerr(ADM)   |
| pointer from last routine call/  | strerror: gets error message . . . .   | strerror(DOS) |
| pointer from last routine call/  | strerror: gets error message . . . .   | strerror(S)   |
| string                           | strftime: format date/time . . . . .   | strftime(S)   |
| /strcpy, strcspn, strdup,        | stricmp: perform operations on/ . . .  | strcat(DOS)   |
| cgets: gets a                    | string . . . . .                       | cgets(DOS)    |
| files  gps: graphical primitive  | string, format of graphical . . . . .  | gps(F)        |
| gets, fgets: gets a              | string from a stream . . . . .         | gets(S)       |
| getclk: gets                     | string from real-time clock . . . . .  | getclk(M)     |
| gets: gets a                     | string from the standard input . . .   | gets(CP)      |
| strstr: finds a                  | string in a string . . . . .           | strstr(DOS)   |
| puts, fputs: puts a              | string on a stream . . . . .           | puts(S)       |
| strdup: performs                 | string operations . . . . .            | string(S)     |
| yes: prints                      | string repeatedly . . . . .            | yes(C)        |
| strftime: format date/time       | string . . . . .                       | strftime(S)   |
| strlen: returns the length of a  | string . . . . .                       | strlen(DOS)   |
| strstr: finds a string in a      | string . . . . .                       | strstr(DOS)   |
| strtod, atof: converts a         | string to a double-precision/ . . . .  | strtod(S)     |
| strtol, atol, atoi: converts     | string to integer . . . . .            | strtol(S)     |
| strset: Sets all characters in a | string to one character . . . . .      | strset(DOS)   |
| cputs: puts a                    | string to the console . . . . .        | cputs(DOS)    |
| strings in an object file        | strings: finds the printable . . . .   | strings(CP)   |
| xstr: extracts                   | strings from C programs . . . . .      | xstr(CP)      |
| strings: finds the printable     | strings in an object file . . . . .    | strings(CP)   |
| compare native language          | strings  nl_strcmp, nl_strcmp: . . .   | nl_strcmp(S)  |
| stricmp: perform operations on   | strings  /strcspn, strdup, . . . . .   | strcat(DOS)   |

|                                 |                                            |                 |
|---------------------------------|--------------------------------------------|-----------------|
| operations on characters in     | strings /strnset: perform . . . . .        | strncat(DOS)    |
| strncoll: handles collation of  | strings /strnxfrm, strcoll, . . . . .      | collation(S)    |
| the order of characters in a    | string strev: reverses . . . . .           | strev(DOS)      |
| relocation bits                 | strip: removes symbols and . . . . .       | strip(CP)       |
| relocation bits                 | strip: removes symbols and . . . . .       | strip(XNX)      |
| string                          | strlen: returns the length of a . . . . .  | strlen(DOS)     |
| characters to lowercase         | strlwr: converts uppercase . . . . .       | strlwr(DOS)     |
| strncpy, strnset: perform/      | strncat, strncmp, strnicmp, . . . . .      | strncat(DOS)    |
| strnset: perform/ strncat,      | strncmp, strnicmp, strncpy, . . . . .      | strncat(DOS)    |
| strxfrm, strnxfrm, strcoll,     | strncoll: handles collation of/ . . . . .  | collation(S)    |
| strncat, strncmp, strnicmp,     | strncpy, strnset: perform/ . . . . .       | strncat(DOS)    |
| perform/ strncat, strncmp,      | strnicmp, strncpy, strnset: . . . . .      | strncat(DOS)    |
| /strncmp, strnicmp, strncpy,    | strnset: perform operations on/ . . . . .  | strncat(DOS)    |
| handles collation of/ strxfrm,  | strnxfrm, strcoll, strncoll: . . . . .     | collation(S)    |
| characters in a string          | strev: reverses the order of . . . . .     | strev(DOS)      |
| string to one charater          | strset: Sets all characters in a . . . . . | strset(DOS)     |
| string                          | strstr: finds a string in a . . . . .      | strstr(DOS)     |
| to a double-precision number    | strtod, atof: converts a string . . . . .  | strtod(S)       |
| string to integer               | strtol, atol, atoi: converts . . . . .     | strtol(S)       |
| mount: mounts a file            | structure . . . . .                        | mount(ADM)      |
| t_alloc: allocate a library     | structure . . . . .                        | t_alloc(NSL)    |
| t_free: free a library          | structure . . . . .                        | t_free(NSL)     |
| umount: dismounts a file        | structure . . . . .                        | umount(ADM)     |
| address of the next heap entry  | structure /returns the . . . . .           | _fheapwalk(DOS) |
| characters to uppercase         | strupr: converts lowercase . . . . .       | strupr(DOS)     |
| strncoll: handles collation of/ | strxfrm, strnxfrm, strcoll, . . . . .      | collation(S)    |
| terminal                        | stty: Sets the options for a . . . . .     | stty(C)         |
| file                            | stune: local tunable parameter . . . . .   | stune(F)        |
| or another user                 | su: makes the user a super-user . . . . .  | su(C)           |
| UUCP rmail:                     | submit: MMDF mail enqueuer . . . . .       | submit(ADM)     |
| /checks for mail which has been | submit remote mail received via . . . . .  | rmail(ADM)      |
| plot: graphics interface        | submitted but not delivered . . . . .      | checkmail(C)    |
| command interface for audit     | subroutines . . . . .                      | plot(S)         |
| files generated by the audit    | subsystem activation,/ auditcmd: . . . . . | auditcmd(ADM)   |
| subsystem: security             | subsystem and /audit collection . . . . .  | auditd(ADM)     |
| produce audit records for       | subsystem component description . . . . .  | subsystem(M)    |
| audit: audit                    | subsystem events dlvr_audit: . . . . .     | dlvr_audit(ADM) |
| component description           | subsystem interface device . . . . .       | audit(ADM)      |
| interface for authorization     | subsystem: security subsystem . . . . .    | subsystem(M)    |
| routines for Subsystems/        | subsystem /administrator . . . . .         | authsh(ADM)     |
| : manipulation routines for     | subsystems : manipulation . . . . .        | subsystems(S)   |
| user data space                 | Subsystems database subsystems . . . . .   | subsystems(S)   |
| /verify machine is              | subyte: stores a character in . . . . .    | subyte(K)       |
| counts blocks in a file         | suitable for security port . . . . .       | check_data(S)   |
| du:                             | sulogin: access single-user mode . . . . . | sulogin(ADM)    |
| ownership quot:                 | sum: calculates checksum and . . . . .     | sum(C)          |
| accounting/ acctcms: command    | summarizes disk usage . . . . .            | du(C)           |
| sync: updates the               | summarizes filesystem . . . . .            | quot(C)         |
| sync: updates the               | summary from per-process . . . . .         | acctcms(ADM)    |
|                                 | super-block . . . . .                      | sync(ADM)       |
|                                 | super-block . . . . .                      | sync(S)         |



|                                  |                                      |               |
|----------------------------------|--------------------------------------|---------------|
| su: makes the user a             | super-user or another user . . . .   | su(C)         |
| if current user is the           | super-user user: determines . . . .  | suser(K)      |
| getgroups: get                   | supplementary group ID's . . . .     | getgroups(S)  |
| initializes native language      | support operation nl_init: . . . .   | nl_init(S)    |
| routes mail locally and over any | supported network mmdf: . . . .      | mmdf(ADM)     |
| terminals: list of               | supported terminals . . . . .        | terminals(M)  |
| /selwakeup: kernel routines      | supporting select(S) . . . . .       | select(K)     |
| keyboard mode or test keyboard   | support kbmode: Set . . . . .        | kbmode(ADM)   |
| /vidumapinit, vidunmap:          | supports video adapter driver/ . . . | video(K)      |
| user is the super-user           | suser: determines if current . . . . | suser(K)      |
| ev_resume: restart a             | suspended queue . . . . .            | ev_resume(S)  |
| signal occurs pause:             | suspends a process until a . . . .   | pause(S)      |
| ev_suspend:                      | suspends an event queue . . . . .    | ev_suspend(S) |
| ev_suspend:                      | suspends an event queue . . . . .    | ev_susp(S)    |
| interval nap:                    | suspends execution for a short . . . | nap(S)        |
| interval sleep:                  | suspends execution for an . . . .    | sleep(C)      |
| interval sleep:                  | suspends execution for an . . . .    | sleep(S)      |
| sleep:                           | suspends processing temporarily . .  | sleep(K)      |
| user data space                  | suword: stores a 32-bit word in . .  | suword(K)     |
|                                  | swab: Swaps bytes . . . . .          | swab(S)       |
| swap:                            | swap administrative interface . . .  | swap(ADM)     |
| swapadd: adds                    | swap area . . . . .                  | swapadd(S)    |
| interface                        | swap: swap administrative . . . .    | swap(ADM)     |
|                                  | swapadd: adds swap area . . . . .    | swapadd(S)    |
| swab:                            | swaps bytes . . . . .                | swab(S)       |
| fdswap:                          | swaps default boot floppy drive . .  | fdswap(ADM)   |
| software modifications to the/   | swconfig: produces a list of the . . | swconfig(C)   |
| file symbol/ ldgetname: retrieve | sxt: pseudo-device driver . . . .    | sxt(M)        |
| object/ /compute the index of a  | symbol name for common object . .    | ldgetname(S)  |
| ldtbread: read an indexed        | symbol table entry of a common . .   | ldtbindex(S)  |
| name for common object file      | symbol table entry of a common/ . .  | ldtbread(S)   |
| syms: common object file         | symbol table entry /symbol . . . .   | ldgetname(S)  |
| file ldtbseek: seek to the       | symbol table format . . . . .        | syms(F)       |
| unistd: file header for          | symbol table of a common object . .  | ldtbseek(S)   |
| sdb: invokes                     | symbolic constants . . . . .         | unistd(F)     |
| strip: removes                   | symbolic debugger . . . . .          | sdb(CP)       |
| strip: removes                   | symbols and relocation bits . . . .  | strip(CP)     |
| table format                     | symbols and relocation bits . . . .  | strip(XNX)    |
|                                  | syms: common object file symbol . .  | syms(F)       |
|                                  | sync: updates the super-block . . .  | sync(ADM)     |
|                                  | sync: updates the super-block . . .  | sync(S)       |
| t_sync:                          | synchronize transport library . . .  | t_sync(NSL)   |
| data segment scenter, sdleave:   | synchronizes access to a shared . .  | scenter(S)    |
| sdgetv, sdwaitv:                 | synchronizes shared data access . .  | sdgetv(S)     |
| select:                          | synchronous I/O multiplexing . . .   | select(S)     |
| command interpreter with C-like  | syntax csh: invokes a shell . . . .  | csh(C)        |
| checks C language usage and      | syntax lint: . . . . .               | lint(CP)      |
| administration utility           | sysadmsh: menu driven system . . .   | sysadmsh(ADM) |
| variables                        | sysconf: get configurable system . . | sysconf(S)    |
| parameters                       | sysdef: output values of tunable . . | sysdef(ADM)   |
| sends system error/ perror,      | sys_errlist, sys_nerr, errno: . . .  | perror(S)     |



|                                  |                                           |               |
|----------------------------------|-------------------------------------------|---------------|
| sysfiles: format of UUCP         | sysfiles file . . . . .                   | sysfiles(F)   |
| sysfiles file                    | sysfiles: format of UUCP . . . . .        | sysfiles(F)   |
| information                      | sysfs: get file system type . . . . .     | sysfs(S)      |
| functions                        | sysi86: machine specific . . . . .        | sysi86(S)     |
| error/ perror, sys_errlist,      | sys_nerr, errno: Sends system             | perror(S)     |
| sar: sar, sa1, sa2, sadc         | system activity report/ . . . . .         | sar(ADM)      |
| cu: calls another UNIX           | system . . . . .                          | cu(C)         |
| procedures brc: brc, bcheckrc    | system initialization . . . . .           | brc(ADM)      |
| mkfs: constructs a file          | system . . . . .                          | mkfs(ADM)     |
| mount: mounts a file             | system . . . . .                          | mount(S)      |
| umount: unmounts a file          | system . . . . .                          | umount(S)     |
| who: lists who is on the         | system . . . . .                          | who(C)        |
| automatically boots the          | system autoboot: . . . . .                | autoboot(ADM) |
| identification file              | systemid: the Micnet system . . . . .     | systemid(F)   |
| the lineprinter spooling         | system lpadmin: configures . . . . .      | lpadmin(ADM)  |
| filesystems and shuts down the   | system /reboot: closes out the            | haltsys(ADM)  |
| commands on a remote UNIX        | system remote: executes . . . . .         | remote(C)     |
| /reboot: closes out the file     | systems and shuts down the/ . . . . .     | haltsys(ADM)  |
| systems: format of UUCP          | systems file . . . . .                    | systems(F)    |
| file                             | systems: format of UUCP Systems           | systems(F)    |
| fsck: checks and repairs file    | systems . . . . .                         | fsck(ADM)     |
| scsi: Small computer             | systems interface . . . . .               | scsi(HW)      |
| labelit: provide labels for file | systems . . . . .                         | labelit(ADM)  |
| checklist: list of file          | systems processed by fsck . . . . .       | checklist(F)  |
| rcp: copies files across UNIX    | systems . . . . .                         | rcp(C)        |
| - mount, unmount multiple file   | systems /mountall, umountall . . . . .    | mountall(ADM) |
| the name of the current UNIX     | system uname: prints . . . . .            | uname(C)      |
| gets name of current UNIX        | system uname: . . . . .                   | uname(S)      |
| device                           | systty: System maintenance . . . . .      | systty(M)     |
| chrtbl: create a ctype locale    | table . . . . .                           | chrtbl(M)     |
| /compute the index of a symbol   | table entry of a common object/ . . . . . | ldtbindex(S)  |
| ldtbread: read an indexed symbol | table entry of a common object/ . . . . . | ldtbread(S)   |
| for common object file symbol    | table entry /symbol name . . . . .        | ldgetname(S)  |
| syms: common object file symbol  | table format . . . . .                    | syms(F)       |
| montbl: create a currency locale | table . . . . .                           | montbl(M)     |
| numtbl: create a numeric locale  | table . . . . .                           | numtbl(M)     |
| ldtbseek: seek to the symbol     | table of a common object file . . . . .   | ldtbseek(S)   |
| setmnt: establishes /etc/mnttab  | table . . . . .                           | setmnt(ADM)   |
| for flaws and creates bad track  | table badtrk: Scans fixed disk . . . . .  | badtrk(ADM)   |
| create a collation locale        | table coltbl: . . . . .                   | coltbl(M)     |
| format of mounted filesystem     | table mnttab: . . . . .                   | mnttab(F)     |
| term: terminal driving           | tables for nroff . . . . .                | term(F)       |
| tables: MMDF Name                | tables: MMDF Name Tables . . . . .        | tables(F)     |
| hdestroy: manages hash search    | tables . . . . .                          | tables(F)     |
| NIC database into channel/domain | tables hsearch, hcreate, . . . . .        | hsearch(S)    |
| tabs: set                        | tables nictable: process . . . . .        | nictable(ADM) |
| request                          | tabs on a terminal . . . . .              | tabs(C)       |
| ctags: creates a                 | tabs: set tabs on a terminal . . . . .    | tabs(C)       |
| a file                           | t_accept: accept a connect . . . . .      | t_accept(NSL) |
|                                  | tags file . . . . .                       | ctags(CP)     |
|                                  | tail: delivers the last part of . . . . . | tail(C)       |

|                                      |                                                   |                 |
|--------------------------------------|---------------------------------------------------|-----------------|
| structure                            | t_alloc: allocate a library . . . . .             | t_alloc(NSL)    |
| performs/ sin, cos,                  | tan, asin, acos, atan, atan2: . . . . .           | trig(S)         |
| functions sinh, cosh,                | tanh: performs hyperbolic . . . . .               | sinh(S)         |
| tape device tapectl: AT&T            | tape control for QIC-24/QIC-02 . . . . .          | tapectl(C)      |
| tape control for QIC-24/QIC-02       | tape device tapectl: AT&T . . . . .               | tapectl(C)      |
| backup: incremental dump             | tape format . . . . .                             | backup(F)       |
| program                              | tape: magnetic tape maintenance . . . . .         | tape(C)         |
| tape: magnetic                       | tape maintenance program . . . . .                | tape(C)         |
| tapedump: dumps magnetic             | tape to output file . . . . .                     | tapedump(C)     |
| QIC-24/QIC-02 tape device            | tapectl: AT&T tape control for . . . . .          | tapectl(C)      |
| output file                          | tapedump: dumps magnetic tape to . . . . .        | tapedump(C)     |
|                                      | tar: archive format . . . . .                     | tar(F)          |
|                                      | tar: archives files . . . . .                     | tar(C)          |
| filelength: returns length of        | target file . . . . .                             | filelength(DOS) |
| transport endpoint                   | t_bind: bind an address to a . . . . .            | t_bind(NSL)     |
| tcsendbreak: line control/           | tcdrain, tcflow, tcflush, . . . . .               | tcflow(S)       |
| line control functions tcdrain,      | tcflow, tcflush, tcsendbreak: . . . . .           | tcflow(S)       |
| control/ tcdrain, tcflow,            | tcflush, tcsendbreak: line . . . . .              | tcflow(S)       |
| functions                            | tgetattr, tcsetattr: state . . . . .              | tcattr(S)       |
| group id functions                   | tsetpgrp, tcsetpgrp: process . . . . .            | tcpggrp(S)      |
| endpoint                             | t_close: close a transport . . . . .              | t_close(NSL)    |
| connection with another/             | t_connect: establish a . . . . .                  | t_connect(NSL)  |
| tcdrain, tcflow, tcflush,            | tcsendbreak: line control/ . . . . .              | tcflow(S)       |
| tcsetattr,                           | tcsetattr: state functions . . . . .              | tcattr(S)       |
| functions tcsetpgrp,                 | tcsetpgrp: process group id . . . . .             | tcpggrp(S)      |
| search trees tsearch, tfind,         | tdelete, twalk: manages binary . . . . .          | tsearch(S)      |
|                                      | tee: creates a tee in a pipe . . . . .            | tee(C)          |
| tee: creates a                       | tee in a pipe . . . . .                           | tee(C)          |
| 4014: paginator for the              | TEKTRONIX 4014 terminal . . . . .                 | 4014(C)         |
| last logins of users and             | teletypes last: indicate . . . . .                | last(C)         |
| temporary file tmpnam,               | tmpnam: creates a name for a . . . . .            | tmpnam(S)       |
| sleep: suspends processing           | temporarily . . . . .                             | sleep(K)        |
| print and/or restrict authorizations | temporarily auths: . . . . .                      | auths(C)        |
| tmpfile: creates a                   | temporary file . . . . .                          | tmpfile(S)      |
| specified cleantmp: remove           | temporary files in directories . . . . .          | cleantmp(ADM)   |
| rmtmp: closes and deletes all        | temporary files in the current/ . . . . .         | rmtmp(DOS)      |
| tmpnam: creates a name for a         | temporary file tmpnam, . . . . .                  | tmpnam(S)       |
| device into/ sptalloc: allocates     | temporary memory or maps a . . . . .              | sptalloc(K)     |
| for nroff                            | term: terminal driving tables . . . . .           | term(F)         |
| terminfo/ captinfo: convert a        | termcap description into a . . . . .              | captinfo(ADM)   |
| data base                            | termcap: terminal capability . . . . .            | termcap(M)      |
| termcap:                             | terminal capability data base . . . . .           | termcap(M)      |
| terminfo:                            | terminal capability data base . . . . .           | terminfo(M)     |
| /putprtcnam: manipulate              | terminal control database entry . . . . .         | getprtcnt(S)    |
| ct: spawn getty to a remote          | terminal . . . . .                                | ct(C)           |
|                                      | terminfo: terminal description database . . . . . | terminfo(S)     |
|                                      | terminfo: terminal description database . . . . . | terminfo(S)     |
| nroff term:                          | terminal driving tables for . . . . .             | term(F)         |
| greek: select                        | terminal filter . . . . .                         | greek(C)        |
| libwindows: windowing                | terminal function library . . . . .               | libwindows(S)   |
| tgetstr, tgoto, tputs: performs      | terminal functions /tgetflag, . . . . .           | termcap(S)      |



|                                  |                                            |                |
|----------------------------------|--------------------------------------------|----------------|
| termio: general                  | terminal interface . . . . .               | termio(M)      |
| termios: pOSIX general           | terminal interface . . . . .               | termios(M)     |
| tty: Special                     | terminal interface . . . . .               | tty(M)         |
| jterm: reset layer of windowing  | terminal . . . . .                         | jterm(C)       |
| dial: establishes an out-going   | terminal line connection . . . . .         | dial(S)        |
| lock: locks a user's             | terminal . . . . .                         | lock(C)        |
|                                  | terminal: login terminal . . . . .         | terminal(HW)   |
| tset: Sets                       | terminal modes . . . . .                   | tset(C)        |
| clear: clears a                  | terminal screen . . . . .                  | clear(C)       |
| gettydefs: Speed and             | terminal settings used by getty . . . . .  | gettydefs(F)   |
| ismpx: return windowing          | terminal state . . . . .                   | ismpx(C)       |
| stty: Sets the options for a     | terminal . . . . .                         | stty(C)        |
| tabs: set tabs on a              | terminal . . . . .                         | tabs(C)        |
| terminal: login                  | terminal . . . . .                         | terminal(HW)   |
| line discipline getty: Sets      | terminal type, modes, speed, and . . . . . | getty(M)       |
| line discipline uugetty: set     | terminal type, modes, speed, and . . . . . | uugetty(ADM)   |
| used between host and windowing  | terminal under layers: protocol . . . . .  | layers(M)      |
| paginator for the TEKTRONIX 4014 | terminal 4014: . . . . .                   | 4014(C)        |
| generates a filename for a       | terminal ctermid: . . . . .                | ctermid(S)     |
| host control of windowing        | terminal jagent: . . . . .                 | jagent(M)      |
| a printer attached to the user's | terminal lprint: print to . . . . .        | lprint(C)      |
| or denies messages sent to a     | terminal mesg: permits . . . . .           | mesg(C)        |
| for the 5620 DMD                 | terminal /object downloader . . . . .      | wtinit(ADM)    |
| enable: turns on                 | terminals and line printers . . . . .      | enable(C)      |
| disable: turns off               | terminals and printers . . . . .           | disable(C)     |
| inittab: alternative login       | terminals file . . . . .                   | inittab(F)     |
| terminals                        | terminals: list of supported . . . . .     | terminals(M)   |
| tty: gets the                    | terminal's name . . . . .                  | tty(C)         |
| terminals: list of supported     | terminals . . . . .                        | terminals(M)   |
| functions of Hewlett-Packard     | terminals hp: handle special . . . . .     | hp(C)          |
| layer multiplexer for windowing  | terminals layers: . . . . .                | layers(C)      |
| of the DASI 450                  | terminal /special functions . . . . .      | 450(C)         |
| of DASI 300 and 300s             | terminals /special functions . . . . .     | 300(C)         |
| tty driver for AT&T windowing    | terminals xt: multiplexed . . . . .        | xt(HW)         |
| isatty: finds the name of a      | terminal ttyname, . . . . .                | ttyname(S)     |
| abort:                           | terminates a process . . . . .             | abort(DOS)     |
| exit, _exit:                     | terminates a process . . . . .             | exit(S)        |
| kill:                            | terminates a process . . . . .             | kill(C)        |
| shutdown:                        | terminates all processing . . . . .        | shutdown(ADM)  |
| exit:                            | terminates the calling process . . . . .   | exit(DOS)      |
| for a child process to stop or   | terminate wait: waits . . . . .            | wait(S)        |
| atexit: calls a process at       | termination . . . . .                      | atexit(DOS)    |
| atexit: calls a process at       | termination . . . . .                      | atexit(S)      |
| for audit subsystem activation,  | termination, /command interface . . . . .  | auditcmd(ADM)  |
| tic:                             | terminfo compiler . . . . .                | tic(C)         |
| tput: queries the                | terminfo database . . . . .                | tput(C)        |
| a termcap description into a     | terminfo description /convert . . . . .    | captoinfo(ADM) |
| infocmp: compare or print out    | terminfo descriptions . . . . .            | infocmp(ADM)   |
| terminfo: format of compiled     | terminfo file . . . . .                    | terminfo(F)    |
| terminfo file                    | terminfo: format of compiled . . . . .     | terminfo(F)    |
| data base                        | terminfo: terminal capability . . . . .    | terminfo(M)    |



|                                  |                                        |                 |
|----------------------------------|----------------------------------------|-----------------|
| database                         | terminfo: terminal description . . .   | terminfo(S)     |
| database                         | terminfo: terminal description . . .   | terminfo(S)     |
| interface                        | termio: general terminal . . . . .     | termio(M)       |
| interface                        | termios: pOSIX general terminal . . .  | termios(M)      |
|                                  | t_error: produce error message . . .   | t_error(NSL)    |
| /isnan: isnand, isnanf:          | test for floating point NaN/ . . . .   | isnan(S)        |
| kbmode: Set keyboard mode or     | test keyboard support . . . . .        | kbmode(ADM)     |
|                                  | test: tests conditions . . . . .       | test(C)         |
| test:                            | tests conditions . . . . .             | test(C)         |
| ed: invokes the                  | text editor . . . . .                  | ed(C)           |
| ex: invokes a                    | text editor . . . . .                  | ex(C)           |
| casual users) edit:              | text editor (variant of ex for . . .   | edit(C)         |
| newform: changes the format of a | text file . . . . .                    | newform(C)      |
| diff: compares two               | text files . . . . .                   | diff(C)         |
| fspec: format specification in   | text files . . . . .                   | fspec(F)        |
| plock: lock process,             | text, or data in memory . . . . .      | plock(S)        |
| binary search trees tsearch,     | tfind, tdelete, twalk: manages . . .   | tsearch(S)      |
|                                  | t_free: free a library structure . . . | t_free(NSL)     |
| tgetstr, tgoto, tputs: performs/ | tgetent, tgetnum, tgetflag, . . . .    | termcap(S)      |
| performs/ tgetent, tgetnum,      | tgetflag, tgetstr, tgoto, tputs: . . . | termcap(S)      |
| service information              | t_getinfo: get protocol-specific . . . | t_getinfo(NSL)  |
| tgoto, tputs: performs/ tgetent, | tgetnum, tgetflag, tgetstr, . . . .    | termcap(S)      |
| state                            | t_getstate: get the current . . . . .  | t_getstate(NSL) |
| tgetent, tgetnum, tgetflag,      | tgetstr, tgoto, tputs: performs/ . . . | termcap(S)      |
| /tgetnum, tgetflag, tgetstr,     | tgoto, tputs: performs terminal/ . . . | termcap(S)      |
|                                  | tic: terminfo compiler . . . . .       | tic(C)          |
| frequency of the system clock in | ticks per second /return the . . . .   | gethz(S)        |
|                                  | time, ftime: gets time and date . . .  | time(S)         |
| clock: the system real-time      | (time of day) clock . . . . .          | clock(F)        |
| sets the system real-time        | (time of day) clock setclock: . . . .  | setclock(ADM)   |
| stime: Sets the                  | time . . . . .                         | stime(S)        |
| executes commands at a later     | time at, batch: . . . . .              | at(C)           |
| time to execute a routine        | timeout, untimeout: schedules a . . .  | timeout(K)      |
| sets up an environment at login  | time profile: . . . . .                | profile(M)      |
| executes commands at specified   | times cron: . . . . .                  | cron(C)         |
| gets process and child process   | times times: . . . . .                 | times(S)        |
| file access and modification     | times utime: Sets . . . . .            | utime(S)        |
| process data and system/         | timex: time a command; report . . . .  | timex(ADM)      |
| time zone                        | timezone: set default system . . . .   | timezone(F)     |
| request                          | t_listen: listen for a connect . . . . | t_listen(NSL)   |
| event on a transport endpoint    | t_look: look at the current . . . . .  | t_look(NSL)     |
| file                             | tmpfile: creates a temporary . . . .   | tmpfile(S)      |
| for a temporary file             | tmpnam, tempnam: creates a name . . .  | tmpnam(S)       |
| /isascii, tolower, toupper,      | toascii: classifies or converts/ . . . | ctype(S)        |
| conv, toupper, tolower,          | toascii: translates characters . . . . | conv(S)         |
| characters conv, toupper,        | tolower, toascii: translates . . . . . | conv(S)         |
| /isgraph, iscntrl, isascii,      | tolower, toupper, toascii:/ . . . .    | ctype(S)        |
| compare shared libraries         | tool chkshlib: . . . . .               | chkshlib(CP)    |
| topology files                   | top, top.next: the Micnet . . . . .    | top(F)          |
| endpoint                         | t_open: establish a transport . . . .  | t_open(NSL)     |
| files top,                       | top.next: the Micnet topology . . . .  | top(F)          |

|                                  |                                        |                   |
|----------------------------------|----------------------------------------|-------------------|
| tsort: Sorts a file              | topologically . . . . .                | tsort(CP)         |
| top, top.next: the Micnet        | topology files . . . . .               | top(F)            |
| transport endpoint               | t_optmgmt: manage options for a        | t_optmgmt(NSL)    |
| acctmrg: merge or add            | total accounting files . . . . .       | acctmrg(ADM)      |
| modification times of a file     | touch: updates access and . . . .      | touch(C)          |
| /iscntrl, isascii, tolower,      | toupper, toascii: classifies or/ . . . | ctype(S)          |
| translates characters conv,      | toupper, tolower, toascii: . . . .     | conv(S)           |
|                                  | tplot: graphics filters . . . . .      | tplot(ADM)        |
| database                         | tput: queries the terminfo . . . .     | tput(C)           |
| /tgetflag, tgetstr, tgoto,       | tputs: performs terminal/ . . . .      | termcap(S)        |
|                                  | tr: translates characters . . . . .    | tr(C)             |
| strace: prints STREAMS           | trace messages . . . . .               | strace(ADM)       |
| ptrace:                          | traces a process . . . . .             | ptrace(S)         |
| and print xt driver packet       | traces xtt: extract . . . . .          | xtt(ADM)          |
| in conjunction with ptrace for   | tracing a child process /used . . .    | paccess(S)        |
| disk for flaws and creates bad   | track table /Scans fixed . . . . .     | badtrk(ADM)       |
| dma_enable: begins DMA           | transfer . . . . .                     | dma_enable(K)     |
| up a DMA controller chip for DMA | transfer dma_param: sets . . . . .     | dma_param(K)      |
| /returns the number of bytes not | transferred during a DMA request       | dma_resid(K)      |
| memory to a user/ db_read:       | transfers data from physical . . .     | db_read(K)        |
| contiguous memory db_write:      | transfers from a user address to .     | db_write(K)       |
| queue files for storing mail in  | transit queue: MMDF . . . . .          | queue(ADM)        |
| trchan:                          | translate character sets . . . . .     | trchan(M)         |
| one format to another            | translate: translates files from .     | translate(C)      |
| conv, toupper, tolower, toascii: | translates characters . . . . .        | conv(S)           |
| tr:                              | translates characters . . . . .        | tr(C)             |
| to another translate:            | translates files from one format .     | translate(C)      |
| setmode: Sets                    | translation mode . . . . .             | setmode(DOS)      |
| decode a binary file for         | transmission via mail uuencode:        | uuencode(C)       |
| encode a binary file for         | transmission via mail uuencode:        | uuencode(C)       |
| t_bind: bind an address to a     | transport endpoint . . . . .           | t_bind(NSL)       |
| t_close: close a                 | transport endpoint . . . . .           | t_close(NSL)      |
| t_open: establish a              | transport endpoint . . . . .           | t_open(NSL)       |
| t_optmgmt: manage options for a  | transport endpoint . . . . .           | t_optmgmt(NSL)    |
| t_unbind: disable a              | transport endpoint . . . . .           | t_unbind(NSL)     |
| look at the current event on a   | transport endpoint t_look: . . . .     | t_look(NSL)       |
| t_sync: synchronize              | transport library . . . . .            | t_sync(NSL)       |
| the scheduler for the uucp file  | transport program uusched: . . .       | uusched(ADM)      |
| a connection with another        | transport user /establish . . . .      | t_connect(NSL)    |
|                                  | trchan: translate character sets . .   | trchan(M)         |
| data sent over a connection      | t_rcv: receive data or expedited . .   | t_rcv(NSL)        |
| confirmation from a connect/     | t_rcvconnect: receive the . . . .      | t_rcvconnect(NSL) |
| from disconnect                  | t_rcvdis: retrieve information . . .   | t_rcvdis(NSL)     |
| an orderly release indication    | t_rcvrel: acknowledge receipt of .     | t_rcvrel(NSL)     |
|                                  | t_rcvudata: receive a data unit . .    | t_rcvudata(NSL)   |
| error indication                 | t_rcvuderr: receive a unit data . .    | t_rcvuderr(NSL)   |
| ftw: walks a file                | tree . . . . .                         | ftw(S)            |
| twalk: manages binary search     | trees tsearch, tfind, tdelete, . . .   | tsearch(S)        |
| acos, atan, atan2: performs      | trigonometric functions /asin, . . .   | trig(S)           |
| i386 - get processor type        | truth value machid: machid, . . .      | machid(C)         |
| with debugging on uutry:         | try to contact remote system . . .     | uutry(ADM)        |



|                                   |                                         |                 |
|-----------------------------------|-----------------------------------------|-----------------|
| manages binary search trees       | tsearch, tfind, tdelete, twalk: . . .   | tsearch(S)      |
|                                   | tset: Sets terminal modes . . .         | tset(C)         |
| data over a connection            | t_snd: send data or expedited . . .     | t_snd(NSL)      |
| disconnect request                | t_snddis: send user-initiated . . .     | t_snddis(NSL)   |
| release                           | t_sndrel: initiate an orderly . . .     | t_sndrel(NSL)   |
|                                   | t_sndudata: send a data unit . . .      | t_sndudata(NSL) |
| topologically                     | tsort: Sorts a file . . .               | tsort(CP)       |
| library                           | t_sync: synchronize transport . . .     | t_sync(NSL)     |
| ttopen, ttout, ttowake,/ tty:     | ttclose, ttin, ttinit, ttiwake, . . .   | tty(K)          |
| ttout, ttowake,/ tty: ttclose,    | ttin, ttinit, ttiwake, ttopen, . . .    | tty(K)          |
| ttowake,/ tty: ttclose, ttin,     | ttinit, ttiwake, ttopen, ttout, . . .   | tty(K)          |
| I/O control commands              | ttiocom: interpret tty driver . . .     | ttiocom(K)      |
| tty: ttclose, ttin, ttinit,       | ttiwake, ttopen, ttout, ttowake,/ . . . | tty(K)          |
| /ttclose, ttin, ttinit, ttiwake,  | ttopen, ttout, ttowake, tthead,/ . . .  | tty(K)          |
| /ttin, ttinit, ttiwake, ttopen,   | ttout, ttowake, tthead, ttrdchk,/ . . . | tty(K)          |
| ttinit, ttiwake, ttopen, ttout,   | ttowake, tthead, ttrdchk,/ /ttin, . . . | tty(K)          |
| /ttopen, ttout, ttowake, tthead,  | ttrdchk, ttrstrt, ttselect,/ . . .      | tty(K)          |
| /ttiwake, ttopen, ttout, ttowake, | tthead, ttrdchk, ttrstrt,/ . . .        | tty(K)          |
| /ttout, ttowake, tthead, ttrdchk, | ttrstrt, ttselect, tttimeo,/ . . .      | tty(K)          |
| /tthead, ttrdchk, ttrstrt,        | ttselect, tttimeo, ttwrite,/ . . .      | tty(K)          |
| /ttrdchk, ttrstrt, ttselect,      | tttimeo, ttwrite, tttxput,/ . . .       | tty(K)          |
| /ttrstrt, ttselect, tttimeo,      | ttwrite, tttxput, ttyflush,/ . . .      | tty(K)          |
| /ttselect, tttimeo, ttwrite,      | tttxput, ttyflush, ttywait: tty/ . . .  | tty(K)          |
| mapchan: format of                | tty device mapping files . . .          | mapchan(F)      |
| mapchan: configure                | tty device mapping . . .                | mapchan(M)      |
| processes raw input data from     | tty device canon: . . .                 | canon(K)        |
| terminals xt: multiplexed         | tty driver for AT&T windowing . . .     | xt(HW)          |
| ttiocom: interpret                | tty driver I/O control commands . . .   | ttiocom(K)      |
| ttxput, ttyflush, ttywait:        | tty driver routines /ttwrite, . . .     | tty(K)          |
|                                   | tty: gets the terminal's name . . .     | tty(C)          |
|                                   | tty: Special terminal interface . . .   | tty(M)          |
| ttiwake, ttopen, ttout,/          | tty: ttclose, ttin, ttinit, . . .       | tty(K)          |
| monochrome, ega, screen:          | tty[01-n], color, . . .                 | screen(HW)      |
| tty2[a-h] , tty2[A-H]:/           | tty1[a-h] , tty1[A-H] , . . .           | serial(HW)      |
| tty2[A-H]: interface/ tty1[a-h]   | tty1[A-H] , tty2[a-h] , . . .           | serial(HW)      |
| tty2[A-H]:/ tty1[a-h] ,           | tty1[A-H] , tty2[a-h] , . . .           | serial(HW)      |
| to/ tty1[a-h] , tty1[A-H] ,       | tty2[a-h] , tty2[A-H]: interface . . .  | serial(HW)      |
| interface/ tty1[a-h] , tty1[A-H]  | tty2[a-h] , tty2[A-H]: . . .            | serial(HW)      |
| /, tty1[A-H] , tty2[a-h] ,        | tty2[A-H]: interface to serial/ . . .   | serial(HW)      |
| ports /, tty1[A-H] , tty2[a-h]    | tty2[A-H]: interface to serial . . .    | serial(HW)      |
| /tttimeo, ttwrite, tttxput,       | ttyflush, ttywait: tty driver/ . . .    | tty(K)          |
| of a terminal                     | ttyname, isatty: finds the name . . .   | ttyname(S)      |
| utmp file of the current user     | ttyslot: finds the slot in the . . .    | ttyslot(S)      |
| /ttwrite, tttxput, ttyflush,      | ttywait: tty driver routines . . .      | tty(K)          |
| mtune:                            | tunable parameter file . . .            | mtune(F)        |
| stune: local                      | tunable parameter file . . .            | stune(F)        |
| attempts to set value of a        | tunable parameter idtune: . . .         | idtune(ADM)     |
| sysdef: output values of          | tunable parameters . . .                | sysdef(ADM)     |
| endpoint                          | t_unbind: disable a transport . . .     | t_unbind(NSL)   |
| /runacct, shutacct, startup,      | turnacct - shell procedures for/ . . .  | acctsh(ADM)     |
| printers disable:                 | turns off terminals and . . .           | disable(C)      |



|                                  |                                             |                 |
|----------------------------------|---------------------------------------------|-----------------|
| accton:                          | turns on accounting . . . . .               | accton(ADM)     |
| printers enable:                 | turns on terminals and line . . . . .       | enable(C)       |
| trees tsearch, tfind, tdelete,   | twalk: manages binary search . . . . .      | tsearch(S)      |
| dtype: determines disk           | type . . . . .                              | dtype(C)        |
| sfsys: local filesystem          | type file . . . . .                         | sfsys(F)        |
| file: determines file            | type . . . . .                              | file(C)         |
| sysfs: get file system           | type information . . . . .                  | sysfs(S)        |
| getty: Sets terminal             | type, modes, speed, and line/ . . . . .     | getty(M)        |
| uugetty: set terminal            | type, modes, speed, and line/ . . . . .     | uugetty(ADM)    |
| machid, i386 - get processor     | type truth value machid: . . . . .          | machid(C)       |
| types                            | types: primitive system data . . . . .      | types(F)        |
| types: primitive system data     | types . . . . .                             | types(F)        |
| file for filesystem              | types mfsys: configuration . . . . .        | mfsys(F)        |
| variable                         | TZ: time zone environment . . . . .         | tz(M)           |
| /localtime, gmtime, asctime,     | tzset: converts date and time to/ . . . . . | ctime(S)        |
|                                  | uadmin: administrative control . . . . .    | uadmin(ADM)     |
|                                  | uadmin: administrative control . . . . .    | uadmin(S)       |
| identity: get or check           | uids or gids from program start . . . . .   | identity(S)     |
| limits                           | ulimit: gets and sets user . . . . .        | ulimit(S)       |
| characters                       | ultoa: converts numbers to . . . . .        | ultoa(DOS)      |
| creation mask                    | umask: Sets and gets file . . . . .         | umask(S)        |
| mask                             | umask: Sets file-creation mode . . . . .    | umask(C)        |
| mask                             | umask: sets the file permission . . . . .   | umask(DOS)      |
| structure                        | umount: dismounts a file . . . . .          | umount(ADM)     |
|                                  | umount: unmounts a filesystem . . . . .     | umount(S)       |
| multiple/ mountall: mountall,    | umountall - mount, unmount . . . . .        | mountall(ADM)   |
| UNIX system                      | uname: gets name of current . . . . .       | uname(S)        |
| current UNIX system              | uname: prints the name of the . . . . .     | uname(C)        |
| uncompress:                      | uncompress a stored file . . . . .          | compress(C)     |
| file                             | uncompress: uncompress a stored . . . . .   | compress(C)     |
| file unget:                      | undoes a previous get of an SCCS . . . . .  | unget(CP)       |
| an SCCS file                     | unget: undoes a previous get of . . . . .   | unget(CP)       |
| into input stream                | ungetc: pushes character back . . . . .     | ungetc(S)       |
| the console buffer               | ungetch: returns a character to . . . . .   | ungetch(DOS)    |
| seed48, lcong48: generates       | uniformly distributed srand48, . . . . .    | drand48(S)      |
| a file                           | uniq: reports repeated lines in . . . . .   | uniq(C)         |
| mktemp: makes a                  | unique filename . . . . .                   | mktemp(S)       |
| constants                        | unistd: file header for symbolic . . . . .  | unistd(F)       |
| t_rcvuderr: receive a            | unit data error indication . . . . .        | t_rcvuderr(NSL) |
| t_rcvudata: receive a data       | unit . . . . .                              | t_rcvudata(NSL) |
| t_sndudata: send a data          | unit . . . . .                              | t_sndudata(NSL) |
|                                  | units: converts units . . . . .             | units(C)        |
| units: converts                  | units . . . . .                             | units(C)        |
| backup: performs                 | UNIX backup functions . . . . .             | backup(ADM)     |
| boot:                            | UNIX boot program . . . . .                 | boot(HW)        |
| intro: introduces                | UNIX commands . . . . .                     | Intro(C)        |
| commands intro: introduces       | UNIX Development System . . . . .           | Intro(CP)       |
| volcopy: make literal copy of    | UNIX filesystem . . . . .                   | volcopy(ADM)    |
| optimal access time dcopy: copy  | UNIX filesystems for . . . . .              | dcopy(ADM)      |
| filesystem backup/ restore: AT&T | UNIX incremental . . . . .                  | restore(ADM)    |
| netutil: administers the         | UNIX network . . . . .                      | netutil(ADM)    |

|                                  |                                           |                |
|----------------------------------|-------------------------------------------|----------------|
| cu: calls another                | UNIX system . . . . .                     | cu(C)          |
| link_unix: builds a new          | UNIX system kernel . . . . .              | link_unix(ADM) |
| /prfdc, prfsnap, prfpr -         | UNIX system profiler . . . . .            | profiler(ADM)  |
| /prfstat, prfdc, prfsnap, prfpr  | UNIX system profiler . . . . .            | profiler(ADM)  |
| uname: gets name of current      | UNIX system . . . . .                     | uname(S)       |
| executes commands on a remote    | UNIX system remote: . . . . .             | remote(C)      |
| rcp: copies files across         | UNIX systems . . . . .                    | rcp(C)         |
| prints the name of the current   | UNIX system uname: . . . . .              | uname(C)       |
| uux: executes command on remote  | UNIX . . . . .                            | uux(C)         |
| uuto, uupick: public             | UNIX-to-UNIX file copy . . . . .          | uuto(C)        |
|                                  | unlink: deletes a file . . . . .          | unlink(DOS)    |
| link: link, unlink: link and     | unlink files and directories . . . . .    | link(ADM)      |
| and directories link: link,      | unlink: link and unlink files . . . . .   | link(ADM)      |
|                                  | unlink: removes directory entry . . . . . | unlink(S)      |
| reading or/ locking: locks or    | unlocks a file region for . . . . .       | locking(S)     |
| /mountall, umountall - mount,    | unmount multiple filesystems . . . . .    | mountall(ADM)  |
| umount:                          | unmounts a filesystem . . . . .           | umount(S)      |
| files pack, pcat,                | unpack: compresses and expands . . . . .  | pack(C)        |
| execute a routine timeout,       | untimeout: schedules a time to . . . . .  | timeout(K)     |
| idinstall: add, delete,          | update, or get device driver/ . . . . .   | idinstall(ADM) |
| performs linear search and       | update lsearch, lfind: . . . . .          | lsearch(S)     |
| times of a file touch:           | updates access and modification . . . . . | touch(C)       |
| of programs make: maintains,     | updates, and regenerates groups . . . . . | make(CP)       |
| sync:                            | updates the super-block . . . . .         | sync(ADM)      |
| sync:                            | updates the super-block . . . . .         | sync(S)        |
| lowercase strlwr: converts       | uppercase characters to . . . . .         | strlwr(DOS)    |
| converts lowercase characters to | uppercase strupr: . . . . .               | strupr(DOS)    |
| about system activity            | uptime: displays information . . . . .    | uptime(C)      |
| lint: checks C language          | usage and syntax . . . . .                | lint(CP)       |
| du: Summarizes disk              | usage . . . . .                           | du(C)          |
| clock: reports CPU time          | used . . . . .                            | clock(S)       |
| keystrokes                       | usemouse: maps mouse input to . . . . .   | usemouse(C)    |
| user su: makes the               | user a super-user or another . . . . .    | su(C)          |
| db_write: transfers from a       | user address to contiguous/ . . . . .     | db_write(K)    |
| data from physical memory to a   | user address db_read: transfers . . . . . | db_read(K)     |
| names id: print                  | user and group IDs and . . . . .          | id(ADM)        |
| id: prints                       | user and group IDs and names . . . . .    | id(C)          |
| setuid, setgid: Sets             | user and group IDs . . . . .              | setuid(S)      |
| /gr_idtoname: map between        | user and group names and IDs . . . . .    | pw_mapping(S)  |
| copyout: copies bytes between    | user and kernel space copyin, . . . . .   | copyin(K)      |
| buffering and/ setvbuf: allow    | user control over stream . . . . .        | setvbuf(DOS)   |
| crontab:                         | user crontab file . . . . .               | crontab(C)     |
| fubyte: gets a character from    | user data space . . . . .                 | fubyte(K)      |
| subyte: stores a character in    | user data space . . . . .                 | subyte(K)      |
| suword: stores a 32-bit word in  | user data space . . . . .                 | suword(K)      |
| gets one 32-bit word from        | user data space fuword: . . . . .         | fuword(K)      |
| /getgid, getegid: gets real      | user, effective user, real/ . . . . .     | getuid(S)      |
| environ: the                     | user environment . . . . .                | environ(M)     |
| hello: Send a message to another | user . . . . .                            | hello(ADM)     |
| getluid: get login               | user ID . . . . .                         | getluid(S)     |
| getpw: gets password for a given | user ID . . . . .                         | getpw(S)       |



|                                   |                                        |                |
|-----------------------------------|----------------------------------------|----------------|
| setluid: set login                | user ID . . . . .                      | setluid(S)     |
| generate disk accounting data by  | user ID diskusg: . . . . .             | diskusg(ADM)   |
| newgrp: logs                      | user into a new group . . . . .        | newgrp(C)      |
| suser: determines if current      | user is the super-user . . . . .       | suser(K)       |
| ulimit: gets and sets             | user limits . . . . .                  | ulimit(S)      |
| logname: finds login name of      | user . . . . .                         | logname(S)     |
| group/ /Gets real user, effective | user, real group, and effective . . .  | getuid(S)      |
| /passc: passes character between  | user space and the kernel . . . . .    | cpass(K)       |
| write: writes to another          | user . . . . .                         | write(C)       |
| gets the login name of the        | user cuserid: . . . . .                | cuserid(S)     |
| with another transport            | user /establish a connection . . . .   | t_connect(NSL) |
| request t_snddis: send            | user-initiated disconnect . . . . .    | t_snddis(NSL)  |
| last: indicate last logins of     | users and teletypes . . . . .          | last(C)        |
| finger: finds information about   | users . . . . .                        | finger(C)      |
| idleout: logs out idle            | users . . . . .                        | idleout(ADM)   |
| lock: locks a                     | user's terminal . . . . .              | lock(C)        |
| to a printer attached to the      | user's terminal lprint: print . . . .  | lprint(C)      |
| wall: writes to all               | users . . . . .                        | wall(ADM)      |
| editor (variant of ex for casual  | users) edit: text . . . . .            | edit(C)        |
| the user a super-user or another  | user su: makes . . . . .               | su(C)          |
| in the utmp file of the current   | user tty slot: finds the slot . . . .  | ttyslot(S)     |
| statistics                        | ustat: gets filesystem . . . . .       | ustat(S)       |
| mscreen: Serial multiscreens      | utility . . . . .                      | mscreen(M)     |
| policy file of the sanitization   | utility purge(C) purge: the . . . .    | purge(F)       |
| at and cron administration        | utility atcronsh: menu driven . . . .  | atcronsh(ADM)  |
| menu driven audit administration  | utility auditsh: . . . . .             | auditsh(ADM)   |
| driven backup administration      | utility backupsh: menu . . . . .       | backupsh(ADM)  |
| lp print service administration   | utility lpsh: menu driven . . . . .    | lpsh(ADM)      |
| driven system administration      | utility sysadmsh: menu . . . . .       | sysadmsh(ADM)  |
| modification times                | utime: Sets file access and . . . . .  | utime(S)       |
| time                              | utime: sets file modification . . . .  | utime(DOS)     |
| utmp, wtmp: formats of            | utmp and wtmp entries . . . . .        | utmp(F)        |
| endutent, utmpname: accesses      | utmp file entry . . . . .              | getut(S)       |
| ttyslot: finds the slot in the    | utmp file of the current user . . . .  | ttyslot(S)     |
| wtmp entries                      | utmp, wtmp: formats of utmp and . .    | utmp(F)        |
| entry endutent,                   | utmpname: accesses utmp file . . . .   | getut(S)       |
| directories and permissions/      | uuchat: dials a modem . . . . .        | dial(ADM)      |
| for work                          | uuchack: check the uucp . . . . .      | uuchack(ADM)   |
| clean-up                          | uucico: Scan the spool directory . . . | uucico(C)      |
| administers                       | uuclean: uucp spool directory . . . .  | uuclean(ADM)   |
| devices: format of                | UUCP control files uuinstall: . . . .  | uuinstall(ADM) |
| file dialcodes: format of         | UUCP devices file . . . . .            | devices(F)     |
| dialers: format of                | UUCP dial-code abbreviations . . . .   | dialcodes(F)   |
| file uuchack: check the           | UUCP Dialers file . . . . .            | dialers(F)     |
| uusched: the scheduler for the    | uucp directories and permissions . .   | uuchack(ADM)   |
| permissions: format of            | uucp file transport program . . . . .  | uusched(ADM)   |
| poll: format of                   | UUCP Permissions file . . . . .        | permissions(F) |
| uulist: converts a XENIX-style    | UUCP Poll file . . . . .               | poll(F)        |
| uuclean:                          | UUCP routing file to/ . . . . .        | uulist(ADM)    |
| control uustat:                   | uucp spool directory clean-up . . . .  | uuclean(ADM)   |
|                                   | uucp status inquiry and job . . . .    | uustat(C)      |



|                                  |                                       |                |
|----------------------------------|---------------------------------------|----------------|
| sysfiles: format of              | UUCP Sysfiles file . . . . .          | sysfiles(F)    |
| systems: format of               | UUCP Systems file . . . . .           | systems(F)     |
| maxuuscheds:                     | UUCP uusched limit file . . . . .     | maxuuscheds(F) |
| maxuuxqts:                       | UUCP uuxqt limit file . . . . .       | maxuuxqts(F)   |
| submit remote mail received via  | UUCP rmail: . . . . .                 | rmail(ADM)     |
| for transmission via mail        | uudecode: decode a binary file . . .  | uuencode(C)    |
| for transmission via mail        | uuencode: encode a binary file . . .  | uuencode(C)    |
| seterror: sets                   | u.u_error with error code . . . . .   | seterror(K)    |
| modes, speed, and line/          | uugetty: set terminal type, . . . . . | uugetty(ADM)   |
| files                            | uuinstall: administers UUCP control   | uuinstall(ADM) |
| UUCP routing file to/            | uulist: converts a XENIX-style . . .  | uulist(ADM)    |
| file copy uuto,                  | uupick: public UNIX-to-UNIX . . .     | uuto(C)        |
| maxuuscheds: UUCP                | uusched limit file . . . . .          | maxuuscheds(F) |
| uucp file transport program      | uusched: the scheduler for the . . .  | uusched(ADM)   |
| job control                      | uustat: uucp status inquiry and . . . | uustat(C)      |
| UNIX-to-UNIX file copy           | uuto, uupick: public . . . . .        | uuto(C)        |
| system with debugging on         | uutry: try to contact remote . . .    | uutry(ADM)     |
| UNIX                             | uux: executes command on remote       | uux(C)         |
| maxuuxqts: UUCP                  | uuxqt limit file . . . . .            | maxuuxqts(F)   |
| val:                             | val: validates an SCCS file . . . . . | val(CP)        |
| assert: helps verify             | validates an SCCS file . . . . .      | val(CP)        |
| abs: returns an integer absolute | validity of program . . . . .         | assert(S)      |
| ceil, fmod: performs absolute    | value . . . . .                       | abs(S)         |
| getenv: gets                     | value, floor, ceiling and/ /fabs, . . | floor(S)       |
| labs: converts to absolute       | value for environment name . . . .    | getenv(S)      |
| cabs: calculates the absolute    | value . . . . .                       | labs(S)        |
| labs: returns the absolute       | value of a complex number . . . .     | cabs(DOS)      |
| /gets and stores the current     | value of a long integer . . . . .     | labs(DOS)      |
| /gets and stores the current     | value of a stream's file/ . . . . .   | fgetpos(DOS)   |
| idntune: attempts to set         | value of a stream's file/ . . . . .   | fgetpos(S)     |
| putenv: changes or adds          | value of a tunable parameter . . .    | idntune(ADM)   |
| true: returns with a zero exit   | value to environment . . . . .        | putenv(S)      |
| returns with a nonzero exit      | value . . . . .                       | true(C)        |
| i386 - get processor type truth  | value false: . . . . .                | false(C)       |
| sysdef: output                   | value machid: machid, . . . . .       | machid(C)      |
| values: machine-dependent        | values: machine-dependent values      | values(M)      |
| the difference between time      | values of tunable parameters . . .    | sysdef(ADM)    |
| the difference between time      | values . . . . .                      | values(M)      |
| /Prints formatted output of a    | values difftime: computes . . . .     | difftime(DOS)  |
| varargs:                         | values difftime: computes . . . .     | difftime(S)    |
| TZ: time zone environment        | varargs argument list . . . . .       | vprintf(S)     |
| sysconf: get configurable system | varargs: variable argument list . .   | varargs(S)     |
| get configurable pathname        | variable . . . . .                    | varargs(S)     |
| edit: text editor                | variables . . . . .                   | tz(M)          |
| vasmapped, vasunbind: virtual/   | variables pathconf: . . . . .         | sysconf(S)     |
| vasunbind: virtual address/ vas: | (variant of ex for casual users) . .  | pathconf(S)    |
| virtual address/ vas: vasbind,   | vas: vasbind, vasmalloc, . . . . .    | edit(C)        |
| vas: vasbind,                    | vasbind, vasmalloc, vasmapped,        | vas(K)         |
| vas: vasbind, vasmalloc,         | vasmapped, vasmapped, vasunbind:      | vas(K)         |
| vas: vasbind, vasmalloc,         | vasmapped, vasunbind: virtual/ . .    | vas(K)         |

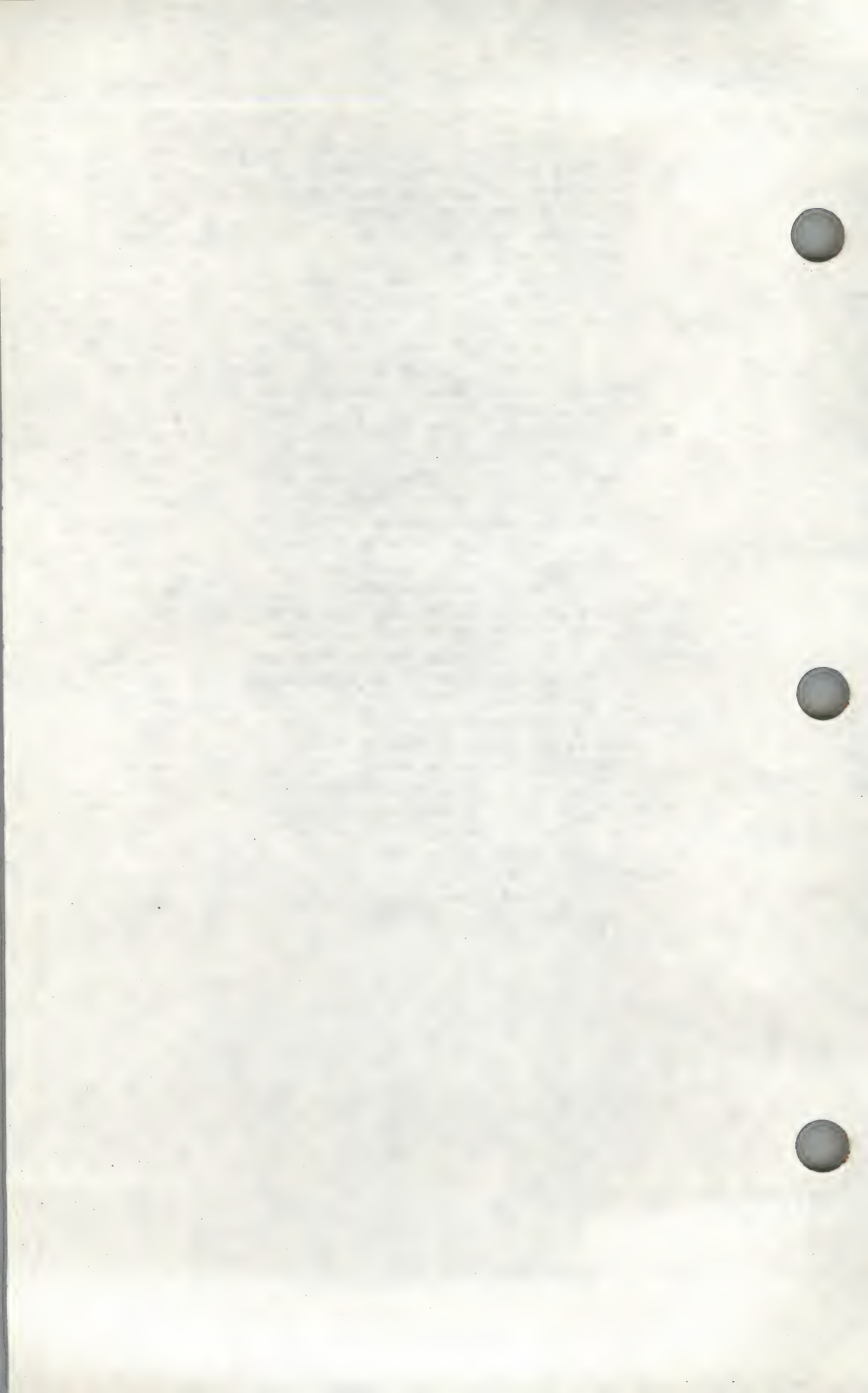
|                                   |                                      |                   |
|-----------------------------------|--------------------------------------|-------------------|
| /vasbind, vasmalloc, vasmapped,   | vasunbind: virtual address space/    | vas(K)            |
|                                   | vc: version control . . . . .        | vc(C)             |
|                                   | vc: version control . . . . .        | vc(CP)            |
| gets option letter from argument  | vector getopt: . . . . .             | getopt(S)         |
| the/ /displays the list of        | vectors currently specified in . . . | vectorsinuse(ADM) |
| of vectors currently specified/   | vectorsinuse: displays the list . .  | vectorsinuse(ADM) |
| display editor vi, view,          | vedit: invokes a screen-oriented     | vi(C)             |
| check_basic_data_structures:      | verify machine is suitable for/ . .  | check_data(S)     |
| assert: helps                     | verify validity of program . . . .   | assert(S)         |
| vc:                               | version control . . . . .            | vc(C)             |
| vc:                               | version control . . . . .            | vc(CP)            |
| red: invokes a restricted         | version of . . . . .                 | ed(C)             |
| scdsdiff: compares two            | versions of an SCCS file . . . . .   | scdsdiff(CP)      |
| formatted output of a/ vprintf,   | vfprintf, vsprintf: prints . . . . . | vprintf(S)        |
| screen-oriented display editor    | vi, view, vedit: invokes a . . . .   | vi(C)             |
| a binary file for transmission    | via mail uuencode: decode . . . .    | uuencode(C)       |
| a binary file for transmission    | via mail uuencode: encode . . . .    | uuencode(C)       |
| submit remote mail received       | via UUCP rmail: . . . . .            | rmail(ADM)        |
| vidresscreen,/ video: DISPLAYED,  | viddoio, vidinitscreen, vidmap,      | video(K)          |
| /vidumapinit, vidunmap: supports  | video adapter driver development .   | video(K)          |
| the font and video mode for a     | video device vidi: Sets . . . . .    | vidi(C)           |
| vidinitscreen, vidmap,/           | video: DISPLAYED, viddoio, . . .     | video(K)          |
| configuration file mvdevice:      | video driver backend . . . . .       | mvdevice(F)       |
| vidi: Sets the font and           | video mode for a video device . .    | vidi(C)           |
| mode for a video device           | vidi: Sets the font and video . . .  | vidi(C)           |
| video: DISPLAYED, viddoio,        | vidinitscreen, vidmap,/ . . . . .    | video(K)          |
| /viddoio, vidinitscreen,          | vidmap, vidresscreen,/ . . . . .     | video(K)          |
| /viddoio, vidinitscreen, vidmap,  | vidresscreen, vidsavscreen,/ . . .   | video(K)          |
| vidunmap:/ /vidmap, vidresscreen, | vidsavscreen, vidumapinit, . . . .   | video(K)          |
| /vidresscreen, vidsavscreen,      | vidumapinit, vidunmap: supports/ .   | video(K)          |
| /vidsavscreen, vidumapinit,       | vidunmap: supports video adapter/    | video(K)          |
| screen-oriented display/ vi,      | view, vedit: invokes a . . . . .     | vi(C)             |
| data paddr: returns               | virtual address pointer to block . . | paddr(K)          |
| /vasmalloc, vasmapped, vasunbind: | virtual address space memory/ . .    | vas(K)            |
| address vtop: convert a           | virtual address to a physical . . .  | vtop(K)           |
| ptok, ktok: converts              | virtual and physical addresses . .   | ptok(K)           |
| vmstat reports                    | virtual memory statistics. . . . .   | vmstat(C)         |
| codeview:                         | visual debugger . . . . .            | codeview(CP)      |
| statistics                        | vmstat: reports virtual memory . .   | vmstat(C)         |
| UNIX filesystem                   | volcopy: make literal copy of . . .  | volcopy(ADM)      |
| filesystem: format of a system    | volume . . . . .                     | filesystem(F)     |
| - format of UNIX system           | volume fs: filesystem . . . . .      | fs(F)             |
| prints formatted output of a/     | vprintf, vfprintf, vsprintf: . . . . | vprintf(S)        |
| output of a/ vprintf, vfprintf,   | vsprintf: prints formatted . . . .   | vprintf(S)        |
| to a physical address             | vtop: convert a virtual address . .  | vtop(K)           |
| who is on the system and what     | w: displays information about . . .  | w(C)              |
| background processes              | wait: awaits completion of . . . .   | wait(C)           |
| iowait:                           | wait for I/O completion . . . . .    | iowait(K)         |
| sigsuspend:                       | wait for signal . . . . .            | sigsuspend(S)     |
| event ev_block:                   | wait until the queue contains an . . | ev_block(S)       |
| to stop or terminate              | wait: waits for a child process . .  | wait(S)           |

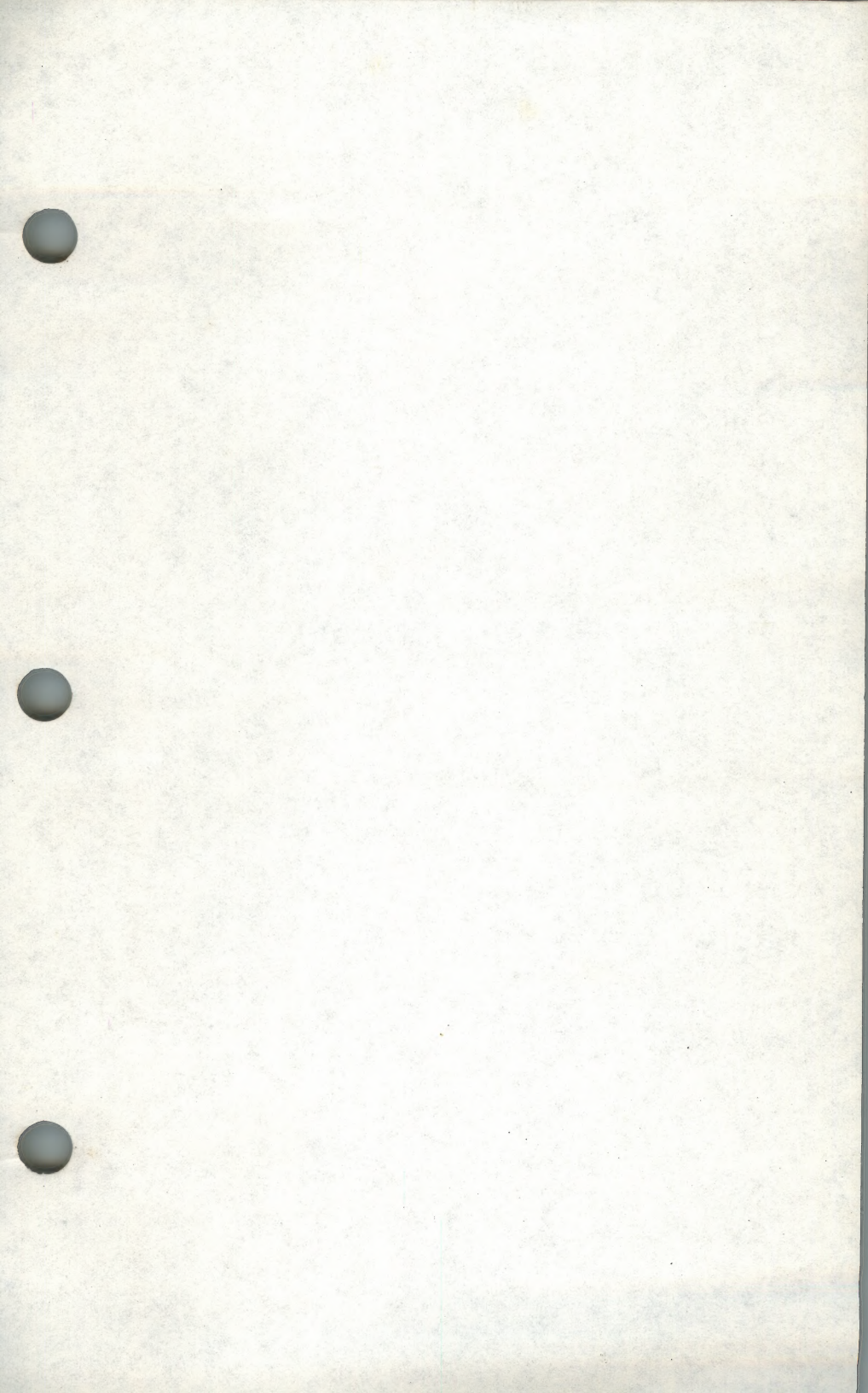


|                                 |                                           |               |
|---------------------------------|-------------------------------------------|---------------|
| sigsem: Signals a process       | waiting on a semaphore . . . . .          | sigsem(S)     |
| stop or terminate wait:         | waits for a child process to . . . . .    | wait(S)       |
| checks access to a resource/    | waitsem, nbwaitsem: awaits and . . . . .  | waitsem(S)    |
| wakeup:                         | wakes up a sleeping process . . . . .     | wakeup(K)     |
| process                         | wakeup: wakes up a sleeping . . . . .     | wakeup(K)     |
| ftw:                            | walks a file tree . . . . .               | ftw(S)        |
|                                 | wall: writes to all users . . . . .       | wall(ADM)     |
| characters                      | wc: counts lines, words and . . . . .     | wc(C)         |
| whodo: determines who is doing  | what . . . . .                            | whodo(C)      |
| what                            | whodo: determines who is doing . . . . .  | whodo(C)      |
| library libwindows:             | windowing terminal function . . . . .     | libwindows(S) |
| jagent: host control of         | windowing terminal . . . . .              | jagent(M)     |
| jterm: reset layer of           | windowing terminal . . . . .              | jterm(C)      |
| ismpx: return                   | windowing terminal state . . . . .        | ismpx(C)      |
| /protocol used between host and | windowing terminal under . . . . .        | layers(M)     |
| layers: layer multiplexer for   | windowing terminals . . . . .             | layers(C)     |
| multiplexed tty driver for AT&T | windowing terminals xt: . . . . .         | xt(HW)        |
| cd: changes                     | working directory . . . . .               | cd(C)         |
| chdir: changes current          | working directory . . . . .               | chdir(DOS)    |
| chdir: changes the              | working directory . . . . .               | chdir(S)      |
| pwd: prints                     | working directory name . . . . .          | pwd(C)        |
| get the pathname of current     | working directory getcwd: . . . . .       | getcwd(S)     |
| scan the spool directory for    | work uucico: . . . . .                    | uucico(C)     |
| fputc, fputchar:                | write a character to a stream . . . . .   | fputc(DOS)    |
| of a/ locking: sets read and    | write permissions for a portion . . . . . | locking(DOS)  |
| putc, putcb, putcbp, putcf:     | write to clists . . . . .                 | putc(K)       |
|                                 | write: writes to a file . . . . .         | write(DOS)    |
|                                 | write: writes to a file . . . . .         | write(S)      |
|                                 | write: writes to another user . . . . .   | write(C)      |
| a physical/ inw, outw: reads,   | writes a 16-bit word from or to . . . . . | inw(K)        |
| physical I/O/ ind, outd: reads, | writes a 32-bit word to a . . . . .       | ind(K)        |
| inb, outb: reads a byte from or | writes a byte to an I/O address . . . . . | inb(K)        |
| outp:                           | writes a byte to an output port . . . . . | outp(DOS)     |
| console putch:                  | writes a character to the . . . . .       | putch(DOS)    |
| putpwent:                       | writes a password file entry . . . . .    | putpwent(S)   |
| /repoutsw, repoutsd: reads and  | writes streams of device data . . . . .   | repins(K)     |
| write:                          | writes to a file . . . . .                | write(DOS)    |
| write:                          | writes to a file . . . . .                | write(S)      |
| wall:                           | writes to all users . . . . .             | wall(ADM)     |
| write:                          | writes to another user . . . . .          | write(C)      |
| fwrite:                         | writes to the output stream . . . . .     | fwrite(DOS)   |
| open: opens file for reading or | writing . . . . .                         | open(S)       |
| a file region for reading or    | writing /Locks or unlocks . . . . .       | locking(S)    |
| opens a file for reading or     | writing open: . . . . .                   | open(DOS)     |
| a file for shared reading and   | writing sopen: opens . . . . .            | sopen(DOS)    |
| the 5620 DMD terminal           | wtinit: object downloader for . . . . .   | wtinit(ADM)   |
| utmp, wtmp: formats of utmp and | wtmp entries . . . . .                    | utmp(F)       |
| entries utmp,                   | wtmp: formats of utmp and wtmp . . . . .  | utmp(F)       |
| accounting/ fwtmp: fwtmp,       | wtmpfix: manipulate connect . . . . .     | fwtmp(ADM)    |
|                                 | x286emul: emulate XENIX 80286 . . . . .   | x286emul(C)   |
| commands                        | xargs: constructs and executes . . . . .  | xargs(C)      |



|                               |                                                 |                         |
|-------------------------------|-------------------------------------------------|-------------------------|
| format                        | xbackup: incremental dump tape                  | . xbackup(F)            |
| incremental filesystem backup | xbackup: performs XENIX                         | . . . xbackup(ADM)      |
| files on a backup archive     | xdumpdir: prints the names of                   | . . . xdumpdir(C)       |
| x286emul: emulate             | XENIX 80286                                     | . . . . . x286emul(C)   |
| assembler asx:                | XENIX 8086/186/286/386                          | . . . . . asx(CP)       |
| masm: invokes the             | XENIX assembler                                 | . . . . . masm(CP)      |
| convert 386 COFF files to     | XENIX format coffconv:                          | . . . . . coffconv(M)   |
| system/ xrestore: invokes     | XENIX incremental file                          | . . . . . xrestore(ADM) |
| filesystem/ xbackup: performs | XENIX incremental                               | . . . . . xbackup(ADM)  |
| xinstall:                     | XENIX installation shell script                 | . . . xinstall(ADM)     |
| dosld:                        | XENIX to MS-DOS cross linker                    | . . . dosld(CP)         |
| mmdfalias: converts           | XENIX-style aliases file to/                    | . . . mmdfalias(ADM)    |
| to/ mnlist: converts a        | XENIX-style Micnet routing file                 | . . . mnlist(ADM)       |
| file to/ uulist: converts a   | XENIX-style UUCP routing                        | . . . uulist(ADM)       |
| shell script                  | xinstall: XENIX installation                    | . . . xinstall(ADM)     |
| entries from files            | xlist, fxlist: gets name list                   | . . . . . xlist(S)      |
| programs                      | xref: cross-references C                        | . . . . . xref(CP)      |
| incremental filesystem/       | xrestore: invokes XENIX                         | . . . . . xrestore(ADM) |
| programs                      | xstr: extracts strings from C                   | . . . . . xstr(CP)      |
| xtt: extract and print        | xt driver packet traces                         | . . . . . xtt(ADM)      |
| xts: extract and print        | xt driver statistics                            | . . . . . xts(ADM)      |
| AT&T windowing terminals      | xt: multiplexed tty driver for                  | . . . . . xt(HW)        |
| channels protocol used by     | xt(HW) driver /multiplexed                      | . . . . . xtproto(M)    |
| protocol used by xt(HW)/      | xtproto: multiplexed channels                   | . . . . . xtproto(M)    |
| statistics                    | xts: extract and print xt driver                | . . . . . xts(ADM)      |
| packet traces                 | xtt: extract and print xt driver                | . . . . . xtt(ADM)      |
| functions                     | bessel, j0, j1, jn, y0, y1, yn: performs Bessel | . . . . . bessel(S)     |
| bessel, j0, j1, jn, y0,       | y1, yn: performs Bessel/                        | . . . . . bessel(S)     |
| compiler-compiler             | yacc: invokes a                                 | . . . . . yacc(CP)      |
|                               | yes: prints string repeatedly                   | . . . . . yes(C)        |
| bessel, j0, j1, jn, y0, y1,   | yn: performs Bessel functions                   | . . . . . bessel(S)     |
|                               | zcat: display a stored file                     | . . . . . compress(C)   |
| true: returns with a          | zero exit value                                 | . . . . . true(C)       |
| sets memory locations to 0    | (zero) bzero:                                   | . . . . . bzero(K)      |
| TZ: time                      | zone environment variable                       | . . . . . tz(M)         |
| set default system time       | zone timezone:                                  | . . . . . timezone(F)   |













514-210-910  
23336